intel®

APPLICATION NOTE

AP-134

October 1981

Asynchronous Communication with the 8274 Multiple-Protocol Serial Controller

Order number: 210311-001

INTRODUCTION

The 8274 Multiprotocol serial controller (MPSC) is a sophisticated dual-channel communications controller that interfaces microprocessor systems to high-speed serial data links (at speeds to 880K bits per second) using synchronous or asynchronous protocols. The 8274 interfaces easily to most common microprocessors (e.g., 8048, 8051, 8085, 8086, and 8088), to DMA controllers such as the 8237 and 8257, and to the 8089 I/O processor. Both MPSC communication channels are completely independent and can operate in a full-duplex communication mode (simultaneous data transmission and reception).

Communication Functions

The 8274 performs many communications-oriented functions, including:

—Converting data bytes from a microprocessor system into a serial bit stream for transmission over the data link to a receiving system.

—Receiving serial bit streams and reconverting the data into parallel data bytes that can easily be processed by the microprocessor system.

—Performing error checking during data transfers. Error checking functions include computing/transmitting error codes (such as parity bits or CRC bytes) and using these codes to check the validity of received data.

—Operating independently of the system processor in a manner designed to reduce the system overhead involved in data transfers.

System Interface

The MPSC system interface is extremely flexible, supporting the following data transfer modes:

1. Polled Mode. The system processor periodically reads (polls) an 8274 status register to determine when a character has been received, when a character is needed for transmission, and when transmission errors are detected.
2. Interrupt Mode. The MPSC interrupts the system processor when a character has been received, when a character is needed for transmission, and when transmission errors are detected.
3. DMA Mode. The MPSC automatically requests data transfers from system memory for both transmit and receive functions by means of two DMA request signals per serial channel. These DMA request signals may be directly interfaced to an 8237 or 8257 DMA controller or to an 8089 I/O processor.
4. WAIT Mode. The MPSC ready signal is used to synchronize processor data transfers by forcing the processor to enter wait states until the 8274 is ready for another data byte. This feature enables the 8274 to interface directly to an 8086 or 8088 processor by means of string I/O instructions for very high-speed data links.

Scope

This application note describes the use of the 8274 in asynchronous communication modes. Asynchronous communication is typically used to transfer data to/from video display terminals, modems, printers, and other low-to-medium-speed peripheral devices. Use of the 8274 in both interrupt-driven and polled system environments is described. Use of the DMA and WAIT modes are not described since these modes are employed mainly in synchronous communication systems where extremely high data rates are common. Programming examples are written in PL/M-86 (Appendix B and Appendix C). PL/M-86 is executed by the iAPX-86 and iAPX-88 processor families. In addition, PL/M-86 is very similar to PL/M-80 (executed by the MCS-80 and MCS-85 processor families). In addition, Appendix D describes a simple application example using an SDK-86 in an iAPX-86/88 environment.

SERIAL-ASYNCHRONOUS DATA LINKS

A serial asynchronous interface is a method of data transmission in which the receiving and transmitting systems need not be synchronized. Instead of transmitting clocking information with the data, locally generated clocks (16, 32 or 64 times as fast as the data transmission rate) are used by the transmitting and receiving systems. When a character of information is sent by the transmitting system, the character data is framed (preceded and followed) by special START and STOP bits. This framing information permits the receiving system to temporarily synchronize with the data transmission. (Refer to Figure 1 during the following discussion of asynchronous data transmission.)

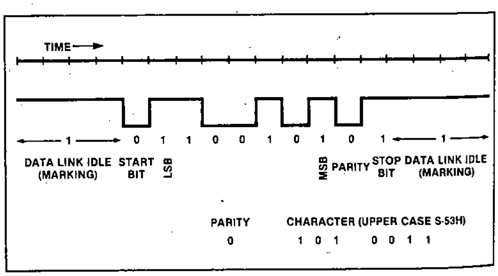


Figure 1. Transmission of a 7-Bit ASCII Character with Even Parity

Normally the data link is in an idle or marking state, continuously transmitting a "mark" (binary 1). When a character is to be sent, the character data bits are immediately preceded by a "space" (binary 0 START bit). The mark-to-space transition informs the receiving system that a character of information will immediately follow the start bit. Figure 1 illustrates the transmission of a 7-bit ASCII character (upper case S) with even parity. Note that the character is transmitted immediately following the start bit. Data bits within the character are transmitted from least-significant to most-significant. The parity bit is transmitted immediately following the character data bits and the STOP framing bit (binary 1) signifies the end of the character.

Asynchronous interfaces are often used with human interface devices such as CRT/keyboard units where the time between data transmissions is extremely variable.

Characters

In asynchronous mode, characters may vary in length from five to eight bits. The character length depends on the coding method used. For example, five-bit characters are used when transmitting Baudot Code, seven-bit characters are required for ASCII data, and eight-bit characters are needed for EBCDIC and binary data. To transmit messages composed of multiple characters, each character is framed and transmitted separately (Figure 2).

This framing method ensures that the receiving system can easily synchronize with the start and stop bits of each character, preventing receiver synchronization errors. In addition, this synchronization method makes both transmitting and receiving systems insensitive to possible time delays between character transmissions.

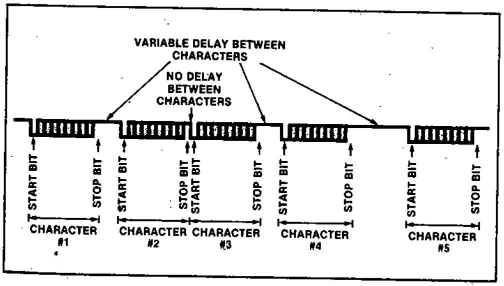


Figure 2. Multiple Character Transmission

Framing

Character framing is accomplished by the START and STOP bits described previously. When the START bit transition (mark-to-space) is detected, the receiving system assumes that a character of data will follow. In order to test this assumption (and isolate noise pulses on the data link), the receiving system waits one-half bit time and samples the data link again. If the link has returned to the marking state, noise is assumed, and the receiver waits for another START bit transition.

When a valid START bit is detected, the receiver samples the data link for each bit of the following character. Character data bits and the parity bit (if required) are sampled at their nominal centers until all required characters are received. Immediately following the data bits, the receiver samples the data link for the STOP bit, indicating the end of the character. Most systems permit specification of 1, 1½, or 2 stop bits.

Timing

The transmitter and receiver in an asynchronous data link arrangement are clocked independently. Normally, each clock is generated locally and the clocks are not synchronized. In fact, each clock may be a slightly different frequency. (In practice, the frequency difference should not exceed a few percent. If the transmitter and receiver clock rates vary substantially, errors will occur because data bits may be incorrectly identified as START or STOP framing bits.) These clocks are designed to operate at 16, 32, or 64 times the communications data rate. These clock speeds allow the receiving device to correctly sample the incoming bit stream.

Serial-interface data rates are measured in bits/second. The term "baud" is used to specify the number of times per second that the transmitted signal level can change states. In general, the baud is not equal to the bit rate. Only when the transmitted signal has two states (electrical levels) is the baud rate equal to the bit rate. Most point-to-point serial data links use RS-232-C, RS-422, or RS-423 electrical interfaces. These specifications call for two electrical signal levels (the baud is equal to the bit rate). Modem interfaces, however, may often have differing bit and baud rates.

While there are generally no limitations on the data transmission rates used in an asynchronous data link, a limited set of rates has been standardized to promote equipment interconnection. These rates vary from 75 bits per second to 38,400 bits per second. Table 1 illustrates typical asynchronous data rates and the associated clock frequencies required for the transmitter and receiver circuits.

Table 1. Communication Data Rates and Associated Transmitter/Receiver Clock Rates

Data Rate (bits/second)	Clock Rate (kHz)		
	X16	X32	X64
75	1.2	2.4	4.8
150	2.4	4.8	9.6
300	4.8	9.6	19.2
600	9.6	19.2	38.4
1200	19.2	38.4	76.8
2400	38.4	76.8	153.6
4800	76.8	153.6	307.2
9600	153.6	307.2	614.2
19200	307.2	614.4	—
38400	614.4	—	—

Parity

In order to detect transmission errors, a parity bit may be added to the character data as it is transferred over the data link. The parity bit is set or cleared to make the total number of "one" bits in the character even (even parity) or odd (odd parity). For example, the letter "A" is represented by the seven-bit ASCII code 1000001 (41H). The transmitted data code (with parity) for this character contains eight bits; 01000001 (41H) for even parity and 11000001 (OC1H) for odd parity. Note that a single bit error changes the parity of the received character and is therefore easily detected. The 8274 supports both odd and even parity checking as well as a parity disable mode to support binary data transfers.

Communication Modes

Serial data transmission between two devices can occur in one of three modes. In the simplex transmission mode, a data link can transmit data in one direction only. In the half-duplex mode, the data link can transmit data in both directions, but not simultaneously. In the full-duplex mode (the most common), the data link can transmit data in both directions simultaneously. The 8274 directly supports the full-duplex mode and will interface to simplex and half-duplex communication data links with appropriate software controls.

BREAK Condition

Asynchronous data links often include a special sequence known as a break condition. A break condition is initiated when the transmitting device forces the data link to a spacing state (binary 0) for an extended length of time (typically 150 milliseconds). Many terminals contain keys to initiate a break sequence. Under software control, the 8274 can initiate a break sequence when transmitting data and detect a break sequence when receiving data.

MPSC SYSTEM INTERFACE

Hardware Environment

The 8274 MPSC interfaces to the system processor over an 8-bit data bus. Each serial I/O channel responds to two I/O or memory addresses as shown in Table 2. In addition, the MPSC supports vectored and daisy-chained interrupts.

The 8274 may be configured for memory-mapped or I/O-mapped operation.

Table 2. 8274 Addressing

$\overline{CS}$	A_1	A_0	Read Operation	Write Operation
0	0	0	Ch. A Data Read	Ch. A Data Write
0	1	0	Ch. A Status Read	Ch. A Command/Parameter
0	0	1	Ch. B Data Read	Ch. B Data Write
0	1	1	Ch. B Status Read	Ch. B Command/Parameter
1	X	X	High Impedence	High Impedence

The 8274-processor hardware interface can be configured in a flexible manner, depending on the operating mode selected—polled, interrupt-driven, DMA, or WAIT. Figure 3 illustrates typical MPSC configurations for use with an 8088 microprocessor in the polled and interrupt-driven modes.

All serial-to-parallel conversion, parallel-to-serial conversion, and parity checking required during asynchronous serial I/O operation is automatically performed by the MPSC.

Operational Interface

Command, parameter, and status information is stored in 22 registers within the MPSC (8 writable registers and 3 readable registers for each channel). These registers are all accessed by means of the command/status ports for each channel. An internal pointer register selects which of the command or status registers will be written or read during a command/status access of an MPSC channel. Figure 4 diagrams the command/status register architecture for each serial channel. In the following discussion, the writable registers will be referred to as WR0 through WR7 and the readable registers will be referred to as RR0 through RR2.

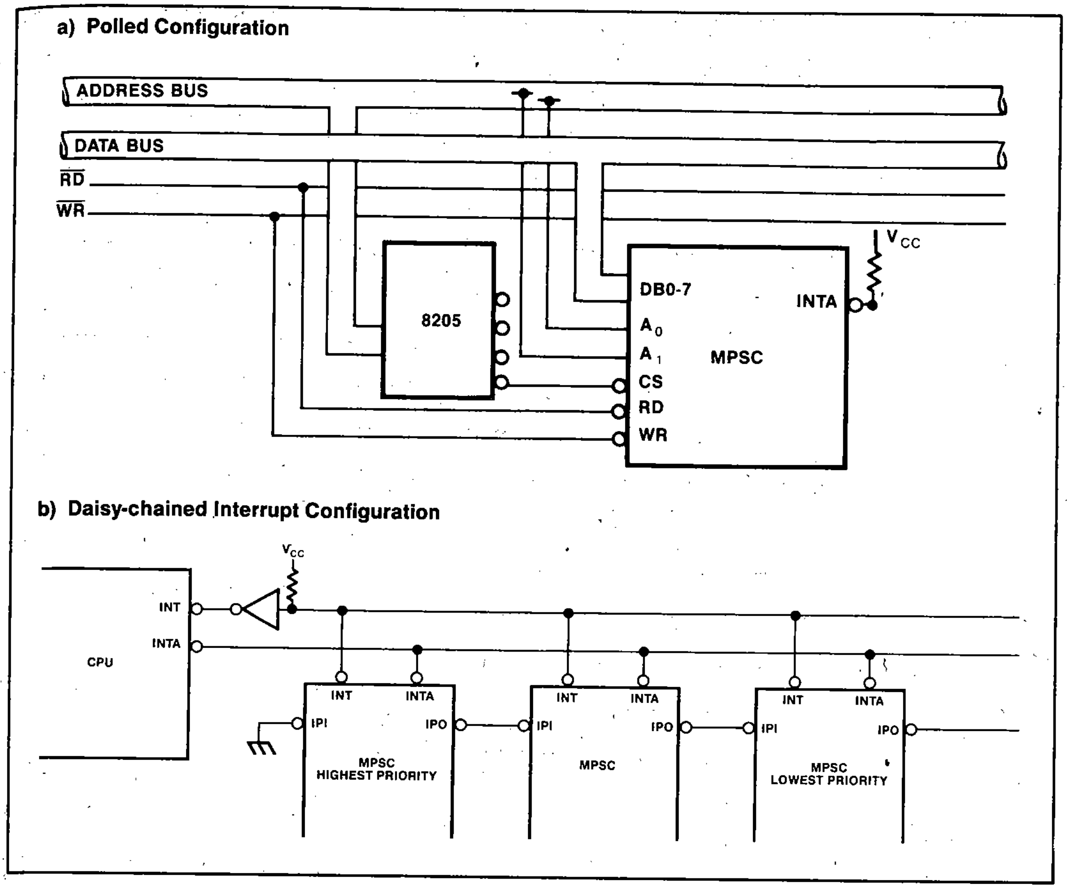


Figure 3. 8274 Hardware Interface for Polled and Interrupt-driven Environments

The least-significant three bits of WR0 are automatically loaded into the pointer register every time WR0 is written. After reset, WR0 is set to zero so that the first write to a command register causes the data to be loaded into WR0 (thereby setting the pointer register). After WR0 is written, the following read or write accesses the register selected by the pointer. The pointer is reset after the read or write operation is completed. In this manner, reading or writing an arbitrary MPSC channel register requires two I/O accesses. The first access is always a write command. This write command is used to set the pointer register. The second access is either a read or a write command; the pointer register (previously set) will ensure that the correct internal register is read or written. After this second access, the pointer register is automatically reset. Note that writing WR0 and reading RR0 does not require presetting of the pointer register.

During initialization and normal MPSC operation, various registers are read and/or written by the system processor. These actions are discussed in detail in the following paragraphs. Note that WR6 and WR7 are not used in the asynchronous communication modes.

RESET

When the 8274 RESET line is activated, both MPSC channels enter the idle state. The serial output lines are forced to the marking state (high) and the modem interface signals (RTS, DTR) are forced high. In addition, the pointer register is set to zero.

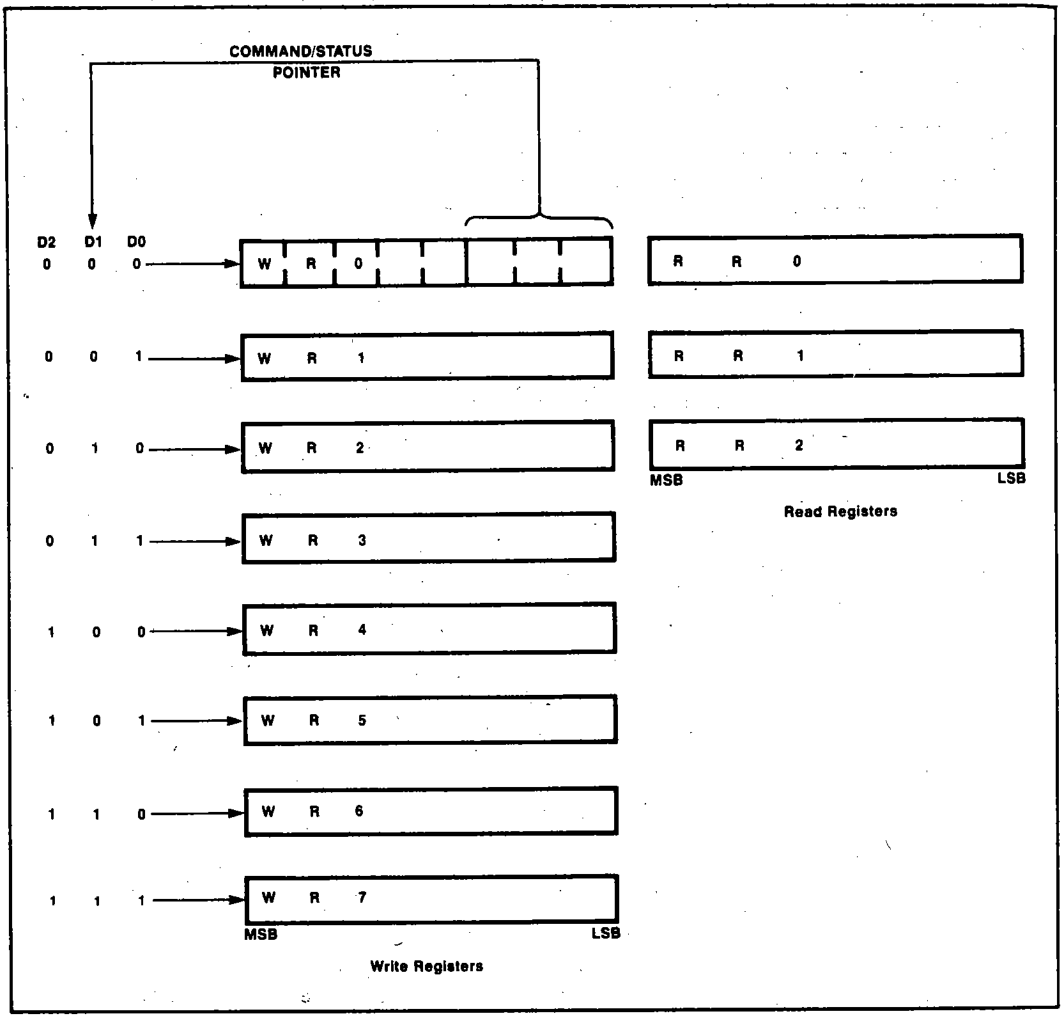


Figure 4. Command/Status Register Architecture (Each Serial Channel)

External/Status Latches

The MPSC continuously monitors the state of four external/status conditions:

1. CTS—clear-to-send input pin.
2. CD—carrier-detect input pin.
3. SYNDET—sync-detect input pin. This pin may be used as a general-purpose input in the asynchronous communication mode.
4. BREAK—a break condition (series of space bits on the receiver input pin).

A change of state in any of these monitored conditions will cause the associated status bit in RR0 (Appendix A) to be latched (and optionally cause an interrupt).

Error Reporting

Three error conditions may be encountered during data reception in the asynchronous mode:

1. Parity. If parity bits are computed and transmitted with each character and the MPSC is set to check parity (bit 0 in WR4 is set), a parity error will occur whenever the number of "1" bits within the character (including the parity bit) does not match the odd/even setting of the parity check flag (bit 1 in WR4).
2. Framing. A framing error will occur if a stop bit is not detected immediately following the parity bit (if parity checking is enabled) or immediately following the most-significant data bit (if parity checking is not enabled).
3. Overrun. If an input character has been assembled but the receiver buffers are full (because the previously received characters have not been read by the system processor), an overrun error will occur. When an overrun error occurs, the input character that has just been received will overwrite the immediately preceding character.

Transmitter/Receiver Initialization

In order to operate in the asynchronous mode, each MPSC channel must be initialized with the following information:

1. Clock Rate. This parameter is specified by bits 6 and 7 of WR4. The clock rate may be set to 16, 32, or 64 times the data-link bit rate. (See Appendix A for WR4 details.)
2. Number of Stop Bits. This parameter is specified by bits 2 and 3 of WR4. The number of stop bits may be set to 1, 1½, or 2. (See Appendix A for WR4 details.)
3. Parity Selection. Parity may be set for odd, even, or no parity by bits 0 and 1 of WR4. (See Appendix A for WR4 details.)
4. Receiver Character Length. This parameter sets the length of received characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 6 and 7 of WR3. (See Appendix A for WR3 details.)
5. Receiver Enable. The serial-channel receiver operation may be enabled or disabled by setting or clearing bit 0 of WR3. (See Appendix A for WR3 details.)
6. Transmitter Character Length. This parameter sets the length of transmitted characters to 5, 6, 7, or 8 bits. This parameter is specified by bits 5 and 6 of WR5. (See Appendix A for WR5 details.) Characters of less than 5 bits in length may be transmitted by setting the transmitted length to five bits (set bits 5 and 6 of WR5 to 1).

   The MPSC then determines the actual number of bits to be transmitted from the character data byte. The bits to be transmitted must be right justified in the data byte, the next three bits must be set to 0 and all remaining bits must be set to 1. The following table illustrates the data formats for transmission of 1 to 5 bits of data:

D7	D6	D5	D4	D3	D2	D1	D0	Number of Bits Transmitted (Character Length)
1	1	1	1	0	0	0	c	1
1	1	1	0	0	0	c	c	2
1	1	0	0	0	c	c	c	3
1	0	0	0	c	c	c	c	4
0	0	0	c	c	c	c	c	5

7. Transmitter Enable. The serial channel transmitter operation may be enabled or disabled by setting or clearing bit 3 of WR5. (See Appendix A for WR5 details.)

For data transmissions via a modem or RS-232-C interface, the following information must also be specified:

1. Request-to-Send/Data-Terminal-Ready. Must be set to indicate status of data terminal equipment. Request-to-send is controlled by bit 1 of WR5 and data terminal ready is controlled by bit 7. (See Appendix A for WR5 details.)
2. Auto Enable. May be set to allow the MPSC to automatically enable the channel transmitter when the clear-to-send signal is active and to automatically enable the receiver when the carrier-detect signal is active. Auto Enable is controlled by bit 5 of WR3. (See Appendix A for WR3 details.)

During initialization, it is desirable to guarantee that the external/status latches reflect the latest interface information. Since up to two state changes are internally stored by the MPSC, at least two Reset External/Status Interrupt commands must be issued. This procedure is most easily accomplished by simply issuing this reset command whenever the pointer register is set during initialization.

An MPSC initialization procedure (MPSCRXINIT) for asynchronous communication is listed in Appendix B. Figure 5 illustrates typical MPSC initialization parameters for use with this procedure.

```
call MPSC$RX$INIT(41,  1,1,0,1,  3,1,1,  3,1,1,0,1);

initializes the 8274 at address 41 as follows:

X16 clock rate                    Enable transmitter and receiver
1 stop bit                        Auto enable set
Odd parity                        DTR and RTS set
8-bit characters (Tx and Rx)      Break transmission disabled
```

Figure 5. Sample 8274 Initialization Procedure for Polled Operation

Polled Operation

In the polled mode, the processor must monitor the MPSC status by testing the appropriate bits in the read register. Data available, status, and error conditions are represented in RR0 and RR1 for channels A and B. An example of MPSC-polled transmitter/receiver routines are given in Appendix B. The following routines are detailed:

1. MPSC$POLL$RCV$CHARACTER—This procedure receives a character from the serial data link. The routine waits until the character-available flag in RR0 has been set. When this flag indicates that a character is available, RR1 is checked for errors (overrun, parity, or framing). If an error is detected, the character in the MPSC receive buffer must be read and discarded and the error routine (RECEIVE$ERROR) is called. If no receive errors have been detected, the character is input from the 8274 data port and returned to the calling program.
   MPSC$POLL$RCV$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the address of a byte variable in which to store the received character (character$ptr).
2. MPSC$POLL$TRAN$CHARACTER—This procedure transmits a character to the serial data link. The routine waits until the transmitter-buffer-empty flag has been set in RR0 before writing the character to the 8274.
   MPSC$POLL$TRAN$CHARACTER requires three parameters—the address of the 8274 channel data port (data$port), the address of the 8274 channel command port (cmd$port), and the character of data that is to be transmitted (character).
3. RECEIVE$ERROR—This procedure processes receiver errors. First, an Error Reset command is written to the affected channel. All additional error processing is dependent on the specific application. For example, the receiving device may immediately request retransmission of the character or wait until a message has been completed.
   RECEIVE$ERROR requires two parameters—the address of the affected 8274 command port (cmd$port) and the error status (status) from 8274 register RR1.

Interrupt-driven Operation

In an interrupt-driven environment, all receiver operations are reported to the system processor by means of interrupts. Once a character has been received and assembled, the MPSC interrupts the system processor. The system processor must then read the character from the MPSC data buffer and clear the current interrupt. During transmission, the system processor starts serial I/O by writing the first character of a message to the MPSC. The MPSC interrupts the system processor whenever the next character is required (i.e., when the transmitter buffer is empty) and the processor responds by writing the next character of the message to the MPSC data port for the appropriate channel.

By using interrupt-driven I/O, the MPSC proceeds independently of the system processor, signalling the processor only when characters are required for transmission, when characters are received from the data link, or when errors occur. In this manner, the system processor may continue execution of other tasks while serial I/O is performed concurrently.

Interrupt Configurations

The 8274 is designed to interface to 8085- and 8086-type processors in much the same manner as the 8259A is designed. When operating in the 8085 mode, the 8274 causes a "call" to a prespecified, interrupt-service routine location. In the 8086 mode, the 8274 presents the processor with a one-byte interrupt-type number. This interrupt-type number is used to "vector" through the 8086 interrupt service table. In either case, the interrupt service address or interrupt-type number is specified during MPSC initialization.

To shorten interrupt latency, the 8274 can be programmed to modify the prespecified interrupt vector so that no software overhead is required to determine the cause of an interrupt. When this "status affects vector" mode is enabled, the following eight interrupts are differentiated automatically by the 8274 hardware:

1. Channel B Transmitter Buffer Empty.
2. Channel B External/Status Transition.
3. Channel B Character Available.
4. Channel B Receive Error.
5. Channel A Transmitter Buffer Empty.
6. Channel A External/Status Transition.
7. Channel A Character Available.
8. Channel A Receive Error.

Interrupt Sources/Priorities

The 8274 has three interrupt sources for each channel:

1. Receiver (RxA, RxB). An interrupt is initiated when a character is available in the receiver buffer or when a receiver error (parity, framing, or overrun) is detected.

2. Transmitter (TxA, TxB): An interrupt is initiated when the transmitter buffer is empty and the 8274 is ready to accept another character for transmission.
3. External/Status (ExTA, ExTB). An interrupt is initiated when one of the external/status conditions (CD, CTS, SYNDET, BREAK) changes state.

The 8274 supports two interrupt priority orderings (selectable during MPSC initialization) as detailed in Appendix A, WR2, CH-A.

Interrupt Initialization

In addition to the initialization parameters required for polled operation, the following parameters must be supplied to the 8274 to specify interrupt operation:

1. Transmit Interrupt Enable. Transmitter-buffer-empty interrupts are separately enabled by bit 1 of WR1. (See Appendix A for WR1 details.)
2. Receive Interrupt Enable. Receiver interrupts are separately enabled in one of three modes: a) interrupt on first received character only and on receive errors (used for message-oriented transmission systems), b) interrupt on all received characters and on receive errors, but do not interrupt on parity errors, and c) interrupt on all received characters and on receive errors (including parity errors). The ability to separately disable parity interrupts can be extremely useful when transmitting messages. Since the parity error bit in RR1 is latched, it will not be reset until an error reset operation is performed. Therefore, the parity error bit will be set if any parity errors were detected in a multicharacter message. If this mode is used, the serial I/O software must poll the parity error bit at the completion of a message and issue an error reset if appropriate. The receiver interrupt mode is controlled by bits 3 and 4 of WR1. (See Appendix A for WR1 details.)
3. External/Status Interrupts. External/Status interrupts can be separately enabled by bit 0 of WR1. (See Appendix A for WR1 details.)
4. Interrupt Vector. An eight-bit interrupt-service routine location (8085) or interrupt type (8086) is specified through WR2 of channel B. (See Appendix A for WR2 details). Table 3 lists interrupt vector addresses generated by the 8274 in the "status affects vector" mode.
5. "Status Affects Vector" Mode. The 8274 will automatically modify the interrupt vector if bit 3 of WR1 is set. (See Appendix A for WR1 details.)
6. System Configuration. Specifies the 8274 data transfer mode. Three configuration modes are available: a) interrupt-driven operation for both channels, b) DMA operation for both channels, and c) DMA operation for channel A, interrupt-driven operation for channel B. The system configuration is specified by means of bits 0 and 1 of WR2 (channel A). (See Appendix A for WR2 details.)
7. Interrupt Priorities. The 8274 permits software specification of receive/transmit priorities by means of bit 2 of WR2 (channel A). (See Appendix A of WR2 details.)
8. Interrupt Mode. Specifies whether the MPSC is to operate in a non-vectored mode (for use with an external interrupt controller), in an 8086-vectored mode, or in an 8085-vectored mode. This parameter is specified through bits 3 and 4 of WR2 (channel A). (See Appendix A for WR2 details.)

Table 3. MPSC-generated Interrupt Vectors in "Status Affects Vector" Mode

V7 V6 V5 V4 V3 V2 V1 V0	V7 V6 V5 V4 V3 V2 V1 V0	Original Vector (specified during Initialization)
8086 Interrupt Type	8085 Interrupt Location	Interrupt Condition
V7 V6 V5 V4 V3 0 0 0	V7 V6 V5 0 0 0 V1 V0	Channel B Transmitter Buffer Empty
V7 V6 V5 V4 V3 0 0 1	V7 V6 V5 0 0 1 V1 V0	Channel B External/Status Change
V7 V6 V5 V4 V3 0 1 0	V7 V6 V5 0 1 0 V1 V0	Channel B Receiver Character Available
V7 V6 V5 V4 V3 0 1 1	V7 V6 V5 0 1 1 V1 V0	Channel B Receive Error
V7 V6 V5 V4 V3 1 0 0	V7 V6 V5 1 0 0 V1 V0	Channel A Transmitter Buffer Empty
V7 V6 V5 V4 V3 1 0 1	V7 V6 V5 1 0 1 V1 V0	Channel A External/Status Change
V7 V6 V5 V4 V3 1 1 0	V7 V6 V5 1 1 0 V1 V0	Channel A Receiver Character Available
V7 V6 V5 V4 V3 1 1 1	V7 V6 V5 1 1 1 V1 V0	Channel A Receive Error

An MPSC interrupt initialization procedure (MPSCINTINIT) is listed in Appendix C.

Interrupt Service Routines

Appendix C lists four interrupt service procedures, a buffer transmission procedure, and a buffer reception procedure that illustrate the use of the 8274 in interrupt-driven environments. Use of these procedures assumes that the 8086/8088 interrupt vector is set to 20H and that channel B is used with the "status affects vector" mode enabled.

1. TRANSMIT$BUFFER—This procedure begins serial transmission of a data buffer. Two parameters are required—a pointer to the buffer (buf$ptr) and the length of the buffer (buf$length). The procedure first sets the global buffer pointer, buffer length, and

initial index for the transmitter-interrupt service routine and initiates transmission by writing the first character of the buffer to the 8274. The procedure then enters a wait loop until the I/O completion status is set by the transmit-interrupt service routine (MPSC$TRANSMIT$CHARACTER$INT).

2. RECEIVE$BUFFER—This procedure inputs a line (terminated by a line feed) from a serial I/O port. Two parameters are required—a pointer to the input buffer (buf$ptr) and a pointer to the buffer length variable (buf$length$ptr). The buffer length will be set by this procedure when the complete line has been input. The procedure first sets the global buffer pointer and initial index for the receiver interrupt service routine. RECEIVE$BUFFER then enters a wait loop until the I/O completion status is set by the receive interrupt routine (MPSC$RECEIVE$-CHARACTER$INT).

3. MPSC$RECEIVE$CHARACTER$INT—This procedure is executed when the MPSC Tx-buffer-empty interrupt is acknowledged. If the current transmit buffer index is less than the buffer length, the next character in the buffer is written to the MPSC data port and the buffer pointer is updated. Otherwise, the transmission complete status is posted.

4. MPSC$RECEIVE$CHARACTER$INT—This procedure is executed when a character has been assembled by the MPSC and the MPSC has issued a character-available interrupt. If no input buffer has been set up by RECEIVE$BUFFER, the character is ignored. If a buffer has been set up, but it is full, a receive overrun error is posted. Otherwise, the received character is read from the MPSC data port and the buffer index is updated. Finally, if the received character is a line feed, the reception complete status is posted.

5. RECEIVE$ERROR$INT—This procedure is executed when a receive error is detected. First, the error conditions are read from RR1 and the character currently in the MPSC receive buffer is read and discarded. Next, an Error Reset command is written to the affected channel. All additional error procession is application dependent.

6. EXTERNAL$STATUS$CHANGE$INT—This procedure is executed when an external status condition change is detected. The status conditions are read from RR0 and a Reset External/Status Interrupt command is issued. Further error processing is application dependent.

DATA LINK INTERFACE

Serial Data Interface

Each serial I/O channel within the 8274 MPSC interfaces to two data link lines—one line for transmitting data and one for receiving data. During transmission, characters are converted from parallel data format (as supplied by the system processor or DMA device) into a serial bit stream (with START and STOP bits) and clocked out on the TxD pin. During reception, a serial bit stream is input on the RxD pin, framing bits are stripped out of the data stream, and the resulting character is converted to parallel data format and passed to the system processor or DMA device.

Data Clocking

As discussed previously, the frequency of data transmission/reception on the data link is controlled by the MPSC clock in conjunction with the programmed clock divider (in register WR4). The 8274 is designed to permit all four serial interface lines (TxD and RxD for each channel) to operate at different data rates. Four clock input pins (TxC and RxC for each channel) are available for this function. Note that the clock rate divider specified in WR4 is used for both RxC and TxC on the appropriate channel; clock rate dividers for each channel are independent.

Modem Control

The following four modem interface signals may be connected to the 8274:

1. Data Terminal Ready (DTR). This interface signal (output by the 8274) is software controlled through bit 7 of WR5. When active, DTR indicates that the data terminal/computer equipment is active and ready to interact with the data communications channel. In addition, this signal prepares the modem for connection to the communication channel and maintains connections previously established (e.g., manual call origination).
2. Request To Send (RTS). This interface signal (output by the 8274) is software controlled through bit 1 of WR5. When active, RTS indicates that the data terminal/computer equipment is ready to transmit data.
3. Clear To Send (CTS). This interface signal (input to the 8274) is supplied by the modem in response to an active RTS signal. CTS indicates that the data terminal/computer equipment is permitted to transmit

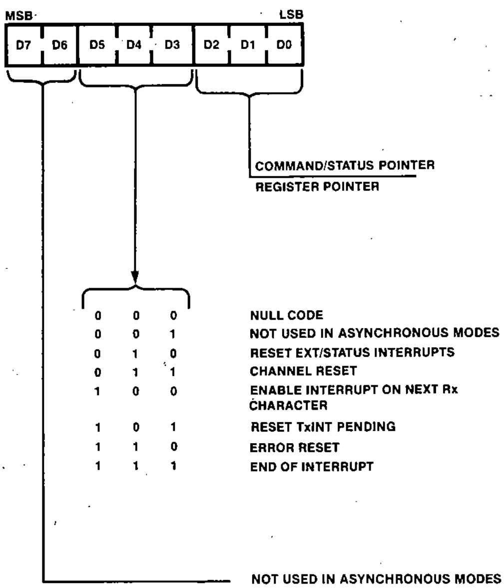
data. The state of CTS is available to the programmer as bit 5 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not transmit data bytes until RTS has been activated. If CTS becomes inactive during transmission of a character, the current character transmission is completed before the transmitter is disabled.

4. Carrier Detect (CD). This interface signal (input to the 8274) is supplied by the modem to indicate that a data carrier signal has been detected and that a valid data signal is present on the RxD line. The state of CD is available to the programmer as bit 3 of RR0. In addition, if the auto enable control is set (bit 5 of WR3), the 8274 will not enable the serial receiver until CD has been activated. If the CD signal becomes inactive during reception of a character, the receiver is disabled, and the partially received character is lost.

In addition to the above modem interface signals, the 8274 SYNDET input pin for channel A may be used as a general-purpose input in the asynchronous communication mode. The status of this signal is available to the programmer as bit 4 of status register RR0.

APPENDIX A
COMMAND/STATUS DETAILS FOR ASYNCHRONOUS COMMUNICATION

Write Register 0 (WR0):



D2,D1,D0 Command/Status Register Pointer bits determine which write-register the next byte is to be written into, or which read-register the next byte is to be read from. After reset, the first byte written into either channel goes into WR0. Following a read or write to any register (except WR0) the pointer will point to WR0.

D5,D4,D3 Command bits determine which of the basic seven commands are to be performed.

Command 0 Null—has no effect.

Command 1 Not used in asynchronous modes.

Command 2 Reset External/Status Interrupts—resets the latched status bits of RR0 and reenables them, allowing interrupts to occur again.

Command 3 Channel Reset—resets the Latched Status bits of RR0, the interrupt prioritization logic and all control registers for the channel. Four extra system clock cycles should be allowed for MPSC reset time before any additional commands or controls are written into the channel.

Command 4 Enable Interrupt on Next Receive Character—if the Interrupt-on-First-Receive Character mode is selected, this command reactivates that mode after each complete message is received to prepare the MPSC for the next message.

Command 5 Reset Transmitter Interrupt Pending—if The Transmit Interrupt mode is selected, the MPSC automatically interrupts data when the transmit buffer becomes empty. When there are no more characters to be sent, issuing this command prevents further transmitter interrupts until the next character has been completely sent.

Command 6 Error Reset—error latches, Parity and Overrun errors in RR1 are reset.

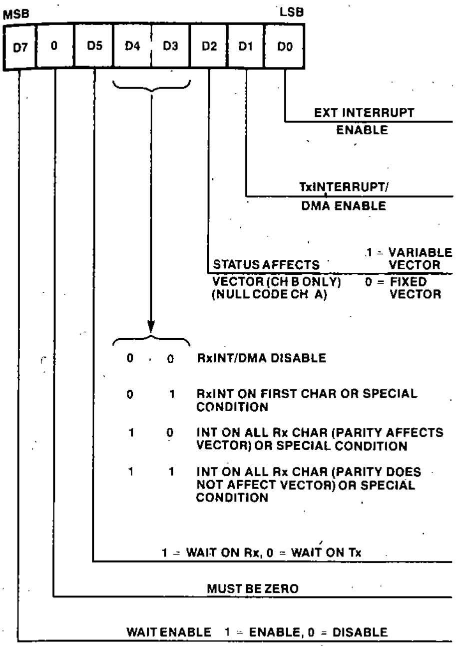
Command 7 End of Interrupt—resets the interrupt-in-service latch of the highest-priority internal device under service.

D0 External/Status Interrupt Enable—allows interrupt to occur as the result of transitions on the $\overline{CD}$, $\overline{CTS}$ or $\overline{SYNDET}$ inputs. Also allows interrupts as the result of a Break/Abort detection and termination, or at the beginning of CRC, or sync character transmission when the Transmit Underrun/EOM latch becomes set.

D1 Transmitter Interrupt/DMA Enable—allows the MPSC to interrupt or request a DMA transfer when the transmitter buffer becomes empty.

D2 Status Affects Vector—(WR1, D2 active in channel B only.) If this bit is not set,

Write Register 1 (WR1):



then the fixed vector, programmed in WR2, is returned from an interrupt acknowledge sequence. If the bit is set, then the vector returned from an interrupt acknowledge is variable as shown in the Interrupt Vector Table.

D4,D3 Receive Interrupt Mode.

0 0 Receive Interrupts/DMA Disabled.

0 1 Receive Interrupt on First Character Only or Special Condition.

1 0 Interrupt on All Receive Characters of Special Condition (Parity Error is a Special Receive Condition).

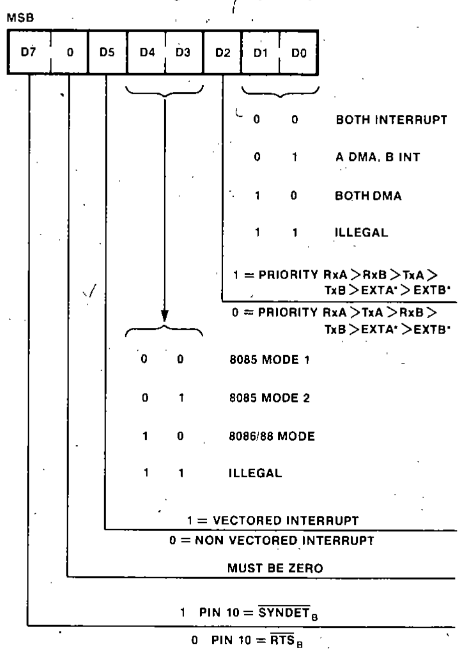
1 1 Interrupt on All Receive Characters or Special Condition (Parity Error is not a Special Receive Condition).

D5 Wait on Receive/$\overline{\text{Transmit}}$—when the following conditions are met, the RDY pin is activated, otherwise it is held in the High-Z state. (Conditions: Interrupt Enabled Mode, Wait Enabled, $\overline{CS}=0$, $A0=0/1$, and $A1=0$). The RDY pin is pulled low when the transmitter buffer is full or the receiver buffer is empty and it is driven High when the transmitter buffer is empty or the receiver buffer is full. The RDY_A and RDY_B may be wired *or* connected since only one signal is active at any one time while the other is in the High Z state.

D6 Must be Zero.

D7 Wait Enable—enables the wait function.

Write Register 2 (WR2): Channel A



*EXTERNAL STATUS INTERRUPT-
ONLY IF EXT INTERRUPT ENABLE (WR1: D0) IS SET

D1,D0 System Configuration—These specify the data transfer from MPSC channels to the CPU, either interrupt or DMA based.

0 0 Channel A and Channel B both use interrupts.

0 1	Channel A uses DMA, Channel Busses interrupt.
1 0	Channel A and Channel B both use DMA.
1 1	Illegal Code.

D2 Priority—this bit specifies the relative priorities of the internal MPSC interrupt/DMA sources.

0	(Highest) RxA, TxA, RxA, RxB, TxBExTA, ExTB (Lowest).
1	(Highest) RxA, RxB, TxA, TxB, ExTA, ExTB (Lowest).

D5,D4,D3 Interrupt Code—specifies the behavior of the MPSC when it receives an interrupt acknowledge sequence from the CPU. (See Interrupt Vector Mode Table).

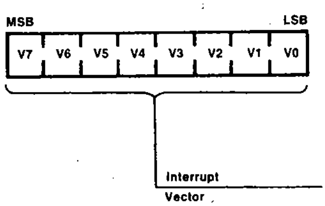
0 X X	Non-vectored interrupts—intended for use with an external interrupt controller such as the 8259A.
1 0 0	8085 Vector Mode 1—intended for use as the primary MPSC in a daisy-chained priority structure.
1 0 1	8085 Vector Mode 2—intended for use as any secondary MPSC in a daisy-chained priority structure.
1 1 0	8086/88 Vector Mode—intended for use as either a primary or secondary in a daisy-chained priority structure.

D6 Must be Zero.

D7

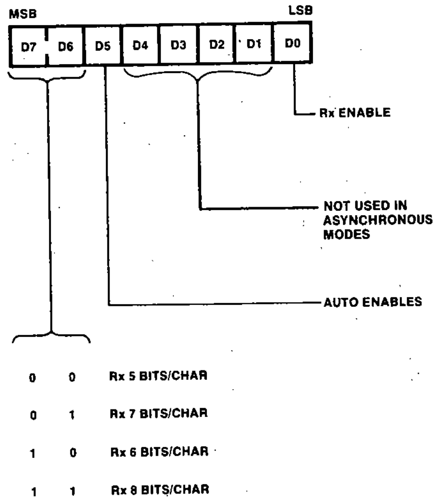
0	Pin 10 = $\overline{RTS_B}$.
1	Pin 10 = $\overline{SYNDET_B}$.

Write Register 2 (WR2): Channel B



D7–D0 Interrupt vector—this register contains the value of the interrupt vector placed on the data bus during acknowledge sequences.

Write Register 3 (WR3):



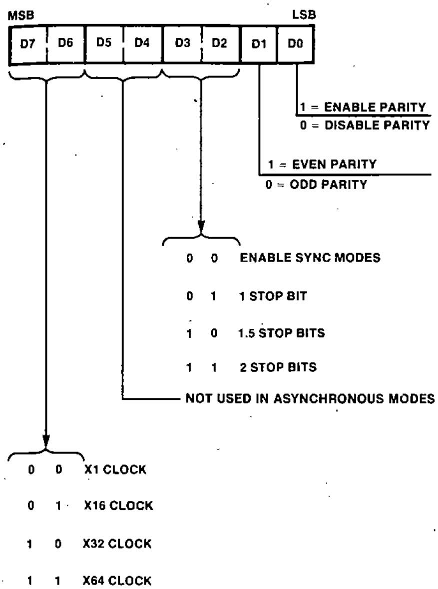
D0 Receiver Enable—A one enables the receiver to begin. This bit should be set only after the receiver has been initialized.

D5 Auto Enables—A one written to this bit causes $\overline{CD}$ to be an automatic enable signal for the receiver and $\overline{CTC}$ to be an automatic enable signal for the transmitter. A zero written to this bit limits the effect of $\overline{CD}$ and $\overline{CTS}$ signals to setting/resetting their corresponding bits in the status register (RR0).

D7,D6 Receiver Character length.

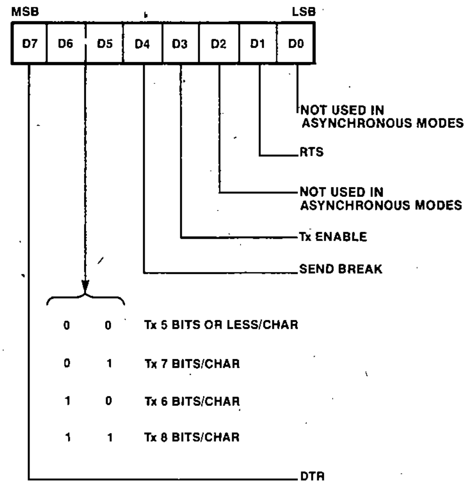
0 0	Receive 5 Data bits/character.
0 1	Receive 7 Data bits/character.
1 0	Receive 6 Data bits/character.
1 1	Receive 8 Data bits/character.

Write Register 4 (WR4):



D0	Parity—a one in this bit causes a parity bit to be added to the programmed number of data bits per character for both the transmitted and received character. If the MPSC is programmed to receive 8 bits per character, the parity bit is not transferred to the microprocessor. With other receiver character lengths, the parity bit is transferred to the microprocessor.
D1	Even/Odd Parity—if parity is enabled, a one in this bit causes the MPSC to transmit and expect even parity, and zero causes it to send and expect odd parity.
D3,D2	Stop Bits.
0 0	Selects synchronous modes.
0 1	Async mode, 1 stop bit/character.
1 0	Async mode, 1½ stop bits/character.
1 1	Async mode, 2 stop bits/character.
D7,D6	Clock mode—selects the clock/data rate multiplier for both the receiver and the transmitter. If the 1x mode is selected, bit synchronization must be done externally.
0 0	Clock rate = Data rate x 1.
0 1	Clock rate = Data rate x 16.
1 0	Clock rate = Data rate x 32.
1 1	Clock rate = Data rate x 64.

Write Register 5 (WR5):



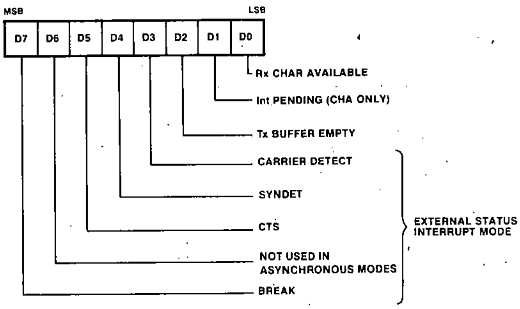
D1	Request to Send—a one in this bit forces the $\overline{RTS}$ pin active (low) and zero in this bit forces the $\overline{RTS}$ pin inactive (high).
D3	Transmitter Enable—a zero in this bit forces a marking state on the transmitter output. If this bit is set to zero during data or sync character transmission, the marking state is entered after the character has been sent. If this bit is set to zero during transmission of a CRC character, sync or flag bits are substituted for the remainder of the CRC bits.
D4	Send Break—a one in this bit forces the transmit data low. A zero in this bit allows normal transmitter operation.
D6,D5	Transmit Character length.
0 0	Transmit 5 or less bits/character.
0 1	Transmit 7 bits/character.
1 0	Transmit 6 bits/character.

1 1	Transmit 8 bits/character.

Bits to be sent must be right justified, least-significant bit first, e.g.:

D7	D6	D5	D4	D3	D2	D1	D0
0	0	B5	B4	B3	B2	B1	B0

Read Register 0 (RR0):



D0 Receive Character Available—this bit is set when the receive FIFO contains data and is reset when the FIFO is empty.

D1 Interrupt Pending—This Interrupt-Pending bit is reset when an E01 command is issued and there is no other interrupt request pending at that time. In vector mode, this bit is set at the falling edge of the second INTA in an INTA cycle for an internal interrupt request. In non-vector mode, this bit is set at the falling edge of RD input after pointer 2 is specified. This bit is always zero in Channel B.

D2 Transmit Buffer Empty—This bit is set whenever the transmit buffer is empty except when CRC characters are being sent in a synchronous mode. This bit is reset when the transmit buffer is loaded. This bit is set after an MPSC reset.

D3 Carrier Detect—This bit contains the state of the CD pin at the time of the last change of any of the External/Status bits ($\overline{\text{CD}}$, CTS, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{\text{CD}}$ pin causes the $\overline{\text{CD}}$ bit to be latched and causes an External/Status interrupt. This bit indicates current state of the $\overline{\text{CD}}$ pin immediately following a Reset External/Status Interrupt command.

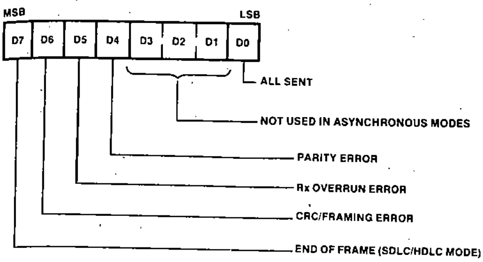
D4 SYNDET—In asynchronous modes, the operation of this bit is similar to the CD status bit, except that it shows the state of the $\overline{\text{SYNDET}}$ input. Any High-to-Low transition on the $\overline{\text{SYNDET}}$ pin sets this bit, and causes an External/Status interrupt (if enabled). The Reset External/Status Interrupt command is issued to clear the interrupt. A Low-to-High transition clears this bit and sets the External/Status interrupt. When the External/Status interrupt is set by the change in state of any other input or condition, this bit shows the inverted state of the $\overline{\text{SYNDET}}$ pin at time of the change. This bit must be read immediately following a Reset External/Status Interrupt command to read the current state of the $\overline{\text{SYNDET}}$ input.

D5 Clear to Send—this bit contains the inverted state of the $\overline{\text{CTS}}$ pin at the time of the last change of any of the External/Status bits (CD, $\overline{\text{CTS}}$, Sync/Hunt, Break/Abort, or Tx Underrun/EOM). Any change of state of the $\overline{\text{CTS}}$ pin causes the $\overline{\text{CTS}}$ bit to be latched and causes an External/Status interrupt. This bit indicates the inverse of the current state of the $\overline{\text{CTS}}$ pin immediately following a Reset External/Status Interrupt command.

D7 Break—in the Asynchronous Receive mode, this bit is set when a Break sequence (null character plus framing error) is detected in the data stream. The External/Status interrupt, if enabled, is set when break is detected. The interrupt service routine must issue the Reset External/Status Interrupt command (WR0, Command 2) to the break detection logic so the Break sequence termination can be recognized.

The Break bit is reset when the termination of the Break sequence is detected in the incoming data stream. The termination of the Break sequence also causes the External/Status interrupt to be set. The Reset External/Status Interrupt command must be issued to enable the break detection logic to look for the next Break sequence. A single, extraneous null character is present in the receiver after the termination of a break; it should be read and discarded.

Read Register 1 (RR1)



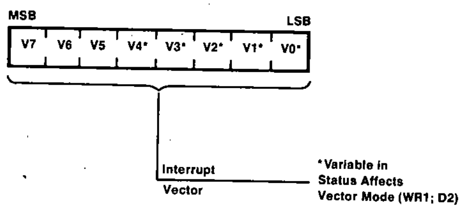
D0 — All sent—this bit is set when all characters have been sent, in asynchronous modes. It is reset when characters are in the transmitter, in asynchronous modes. In synchronous modes, this bit is always set.

D4 — Parity Error—if parity is enabled, this bit is set for received characters whose parity does not match the programmed sense (Even/Odd). This bit is latched. Once an error occurs, it remains set until the Error Reset command is written.

D5 — Receive Overrun Error—this bit indicates that the receive FIFO has been overloaded by the receiver. The last character in the FIFO is overwritten and flagged with this error. Once the overwritten character is read, this error condition is latched until reset by the Error Reset command. If the MPSC is in the "status affects vector" mode, the overrun causes a special Receive Error Vector.

D6 — Framing Error—in async modes, a one in this bit indicates a receive framing error. It can be reset by issuing an Error Reset command.

Read Register 2 (RR2):



RR2 — Channel B

D7–D0 — Interrupt vector—contains the interrupt vector programmed into WR2. If the "status affects vector" mode is selected, it contains the modified vector. (See WR2.) RR2 contains the modified vector for the highest priority interrupt pending. If no interrupts are pending, the variable bits in the vector are set to one.

APPENDIX B
MPSC-POLLED TRANSMIT/RECEIVE CHARACTER ROUTINES

```
MPSC$RX$INIT: procedure (cmd$port,
                         clock$rate,stop$bits,parity$type,parity$enable,
                         rx$char$length,rx$enable,auto$enable,
                         tx$char$length,tx$enable,dtr,brk,rts);

  declare cmd$port       byte,
        clock$rate       byte,
        stop$bits        byte,
        parity$type      byte,
        parity$enable    byte,
        rx$char$length   byte,
        rx$enable        byte,
        auto$enable      byte,
        tx$char$length   byte,
        tx$enable        byte,
        dtr              byte,
        brk              byte,
        rts              byte;


  output(cmd$port)=30H;            /* channel reset */

  output(cmd$port)=14H;            /* point to WR4 */
  /* set clock rate, stop bits, and parity information */
  output(cmd$port)=shl(clock$rate,6) or shl(stop$bits,2) or shl(parity$type,1)
                 or parity$enable;

  output(cmd$port)=13H;            /* point to WR3 */
  /* set up receiver parameters */
  output(cmd$port)=shl(rx$char$length,6) or rx$enable or shl(auto$enable,5);

  output(cmd$port)=15H;            /* point to WR5 */
  /* set up transmitter parameters */
  output(cmd$port)=shl(tx$char$length,5) or shl(tx$enable,3) or shl(dtr,7)
                 or shl(brk,4) or shl(rts,1);

  end MPSC$RX$INIT;
```

```
MPSC$POLL$RCV$CHARACTER: procedure(data$port,cmd$port,character$ptr) byte;

  declare data$port     byte,
          cmd$port      byte,
          character$ptr pointer,
          character     based character$ptr byte,
          status        byte;

  declare char$avail    literally '1',
          rcv$error     literally '70H';

  /* wait for input character ready */
  while (input(cmd$port) and char$avail) <> 0 do; end;

  /* check for errors in received character */
  output(cmd$port)=1;                           /* point to RR1 */
  if (status:=input(cmd$port) and rcv$error)
    then do;
      character=input(data$port);               /* read character to clear MPSC */
      call RECEIVE$ERROR(cmd$port,status);      /* clear receiver errors */
      return 0;                                 /* error return - no character avail */
      end;
    else do;
      character=input(data$port);
      return 0FFH;                              /* good return - character avail */
      end;

end MPSC$POLL$RCV$CHARACTER;


MPSC$POLL$TRAN$CHARACTER: procedure(data$port,cmd$port,character);

  declare data$port     byte,
          cmd$port      byte,
          character     byte;

  declare tx$buffer$empty literally '4';

  /* wait for transmitter buffer empty */
  while not (input(cmd$port) and tx$buffer$empty) do; end;

  /* output character */
  output(data$port)=character;

end MPSC$POLL$TRAN$CHARACTER;


RECEIVE$ERROR: procedure(cmd$port,status);

  declare cmd$port      byte,
          status        byte;

  output(cmd$port)=30H;              /* error reset */

  /* *** other application dependent
         error processing should be placed here    *** */

end RECEIVE$ERROR;
```

```
TRANSMIT$BUFFER: procedure(buf$ptr,buf$length)

  declare
    buf$ptr        pointer,
    buf$length     byte;

  /* set up transmit buffer pointer and buffer length in global variables for
     interrupt service */
  tx$buffer$ptr=buf$ptr;
  transmit$length=buf$length;

  transmit$status=not$complete;            /* setup status for not complete */
  output(data$port)=transmit$buffer(0);    /* transmit first character */
  transmit$index=1;                        /* first character transmitted */

  /* wait until transmission complete or error detected */
  while transmit$status = not$complete do; end;
  if transmit$status <> complete
    then return false;
    else return true;

end TRANSMIT$BUFFER;


RECEIVE$BUFFER: procedure (buf$ptr,buf$length$ptr);

  declare
    buf$ptr         pointer,
    buf$length$ptr  pointer,
    buf$length      based buf$length$ptr byte;

  /* set up receive buffer pointer in global variable for interrupt service */
  rx$buffer$ptr=buf$ptr;
  receive$index=0;

  receive$status=not$complete;          /* set status to not complete */
  /* wait until buffer received */
  while receive$status = not$complete do; end;
  buf$length=receive$length;
  if receive$status = complete
    then return true;
    else return false;

end RECEIVE$BUFFER;
```

```
MPSC$RECEIVE$CHARACTER$INT: procedure interrupt 22H;

  /* ignore input if no open buffer */
  if receive$status <> not$complete then return;

  /* check for receive buffer overrun */
  if receive$index = 128
    then receive$status=overrun;
    else do;
      /* read character from MPSC and place in buffer - note that the
         parity of the character must be masked off during this step if
         the character is less than 8 bits (e.g., ASCII)  */
      receive$buffer(receive$index),character=input(data$port) and 7FH;
      receive$index=receive$index+1;      /* update receive buffer index */

      /* check for line feed to end line */
      if character = line$feed
        then do; receive$length=receive$index; receive$status=complete; end;
    end;

end MPSC$RECEIVE$CHARACTER$INT;


MPSC$TRANSMIT$CHARACTER$INT: procedure interrupt 20H;

  /* check for more characters to transfer */
  if transmit$index < transmit$length
    then do;
      /* write next character from buffer to MPSC */
      output(data$port)=transmit$buffer(transmit$index);
      transmit$index=transmit$index+1;     /* update transmit buffer index */
    end;
    else transmit$status=complete;

end MPSC$TRANSMIT$CHARACTER$INT;


RECEIVE$ERROR$INT: procedure interrupt 23H;

  declare
    temp                    byte;     /* temporary character storage */

  output(cmd$port)=1;                 /* point to RR1 */
  receive$status=input(cmd$port);
  temp=input(data$port);              /* discard character */
  output(cmd$port)=error$reset;       /* send error reset */

  /* *** other application dependent
         error processing should be placed here   *** */

end RECEIVE$ERROR$INT;


EXTERNAL$STATUS$CHANGE$INT: procedure interrupt 21H;

  transmit$status=input(cmd$port)      /* input status change information */
  output(cmd$port)=reset$ext$status;

  /* *** other application dependent
         error processing should be placed here   *** */

end EXTERNAL$STATUS$CHANGE$INT;
```

APPENDIX C
INTERRUPT-DRIVEN TRANSMIT/RECEIVE SOFTWARE

```
declare
  /* global variables for buffer manipulation */

  rx$buffer$ptr           pointer,           /* pointer to receive buffer */
  receive$buffer based rx$buffer$ptr(128) byte,
  receive$status          byte initial(0),   /* indicates receive buffer status */
  receive$index           byte,              /* current index into receive buffer */
  receive$length          byte,              /* length of final receive buffer */

  tx$buffer$ptr           pointer,           /* pointer to transmit buffer */
  transmit$buffer based tx$buffer$ptr(128) byte,
  transmit$status         byte initial(0),   /* indicates transmit buffer status */
  transmit$index          byte,              /* current index into transmit buffer */
  transmit$length         byte,              /* length of buffer to be transmitted */

  cmd$port                literally '43H',
  data$port               literally '41H',
  a$cmd$port              literally '42H',
  b$cmd$port              literally '43H',
  line$feed               literally '0AH',
  not$complete            literally '0',
  complete                literally '0FFH',
  overrun                 literally '1',

  channel$reset           literally '18H',
  error$reset             literally '30H',
  reset$ext$status        literally '10H';
```

```
MPSC$INT$INIT: procedure (clock$rate,stop$bits,parity$type,parity$enable,
                          rx$char$length,rx$enable,auto$enable,
                          tx$char$length,tx$enable,dtr,brk,rts,
                          ext$en,tx$en,rx$en,stat$affects$vector,
                          config,priority,vector$int$mode,int$vector);

  declare
        clock$rate          byte,        /* 2-bit code for clock rate divisor */
        stop$bits           byte,        /* 2-bit code for number of stop bits */
        parity$type         byte,        /* 1-bit parity type */
        parity$enable       byte,        /* 1-bit parity enable */
        rx$char$length      byte,        /* 2-bit receive character length */
        rx$enable           byte,        /* 1-bit receiver enable */
        auto$enable         byte,        /* 1-bit auto enable flag */
        tx$char$length      byte,        /* 2-bit transmit character length */
        tx$enable           byte,        /* 1-bit transmitter enable */
        dtr                 byte,        /* 1-bit status of DTR pin */
        brk                 byte,        /* 1-bit data link break enable */
        rts                 byte,        /* 1-bit status of RTS pin */
        ext$en              byte,        /* 1-bit external/status enable */
        tx$en               byte,        /* 1-bit Tx interrupt enable */
        rx$en               byte,        /* 2-bit Rx interrupt enable/mode */
        stat$aff$vector     byte,        /* 1-bit status affects vector flag */
        config              byte,        /* 2-bit system config - int/DMA */
        priority            byte,        /* 1-bit priority flag */
        vector$int$mode     byte,        /* 3-bit interrupt mode code */
        int$vector          byte;        /* 8-bit interrupt type code */


  output(b$cmd$port)=channel$reset;   /* channel reset */

  output(b$cmd$port)=14H;             /* point to WR4 */
  /* set clock rate, stop bits, and parity information */
  output(b$cmd$port)=shl(clock$rate,6) or shl(stop$bits,2) or shl(parity$type,1)
                  or parity$enable;

  output(b$cmd$port)=13H;             /* point to WR3 */
  /* set up receiver parameters */
  output(b$cmd$port)=shl(rx$char$length,6) or rx$enable or shl(auto$enable,5);

  output(b$cmd$port)=15H;          /* point to WR5 */
  /* set up transmitter parameters */
  output(b$cmd$port)=shl(tx$char$length,5) or shl(tx$enable,3) or shl(dtr,7)
                or shl(brk,4) or shl(rts,1);

  output(b$cmd$port)=12H;               /* point to WR2 */
  /* set up interrupt vector */
  output(b$cmd$port)=int$vector;

  output(a$cmd$port)=12H;               /* point to WR2, channel A */
  /* set up interrupt modes */
  output(a$cmd$port)=shl(vector$int$mode,3) or shl(priority,2) or config;

  output(b$cmd$port)=11H;               /* point to WR1 */
  /* set up interrupt enables */
  output(b$cmd$port)=shl(rx$en,3) or shl(stat$aff$vector,2) or shl(tx$en,1)
                  or ext$en;

  end MPSC$INT$INIT;
```

APPENDIX D
APPLICATION EXAMPLE USING SDK-86

This application example shows the 8274 in a simple iAPX-86/88 system. The 8274 controls two separate asynchronous channels using its internal interrupt controller to request all data transfers. The 8274 driver software is described which transmits and receives data buffers provided by the CPU. Also, status registers are maintained in system memory to allow the CPU to monitor progress of the buffers and error conditions.

THE HARDWARE INTERFACE

Nothing could be easier than the hardware design of an interrupt-driven 8274 system. Simply connect the data bus lines, a few bus control lines, supply a timing clock for baud rate and, voila, it's done! For this example, the ubiquitous SDK-86 is used as the host CPU system. The 8274 interface is constructed on the wire-wrap area provided. While discussing the hardware interface, please refer to Diagram 1.

Placing the 8274 on the lower 8 bits of the 8086 data bus allows byte-wide data transfers at even I/O addresses. For simplicity, the 8274's CS/ input is generated by combining the M-IO/ select line with address line A7 via a 7432. This places the 8274 address range in multiple spots within the 8086 I/O address space. (While fine for this example, a more complete address decoding is recommended for actual prototype systems.) The 8086's A1 and A2 address lines are connected to the A0 and A1 8274 register select inputs respectively. Although other port assignments are possible because of the overlapping address spaces, the following I/O port assignments are used in this example:

Port Function	I/O Address
Data channel A	0000H
Command/status A	0002H
Data channel B	0004H
Command/status B	0006H

To connect the 8274's interrupt controller into the system an inverter and pull-up resistor are needed to convert the 8274's active-low, interrupt-request output, IRQ, into the correct polarity for the 8086's INTR interrupt input. The 8274 recognizes interrupt-acknowledge bus cycles by connecting the INTA (INTerrupt Acknowledge) lines of the 8274 and 8086 together.

The 8274 ReaD and WRite lines directly connect to the respective 8086 lines. The RESET line requires an inverter. The system clock for the 8274 is provided by the PCLK (peripheral clock) output of the 8284A clock generator.

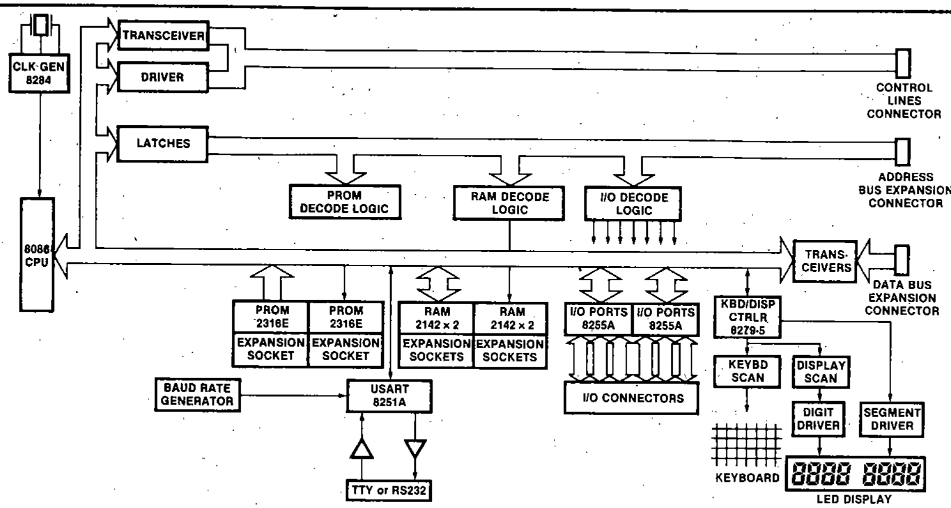
On the 8274's serial side, traditional 1488 and 1489 RS-232 drivers and receivers are used for the serial interface. The onboard baud rate generator supplies the channel baud rate timing. In this example, both sides of both channels operate at the same baud rate although this certainly is not a requirement. (On the SDK-86, the baud rate selection is hard-wired thru jumpers. A more flexible approach would be to incorporate an 8253 Programmable Interval Timer to allow software-configurable baud rate selection.)

That's all there is to it. This hardware interface is completely general-purpose and supports all of the 8274 features except the DMA data transfer mode which requires an external DMA controller. Now let's look at the software interface.

SOFTWARE INTERFACE

In this example, it is assumed that the 8086 has better things to do rather than continuously run a serial channel. Presenting the software as a group of callable procedures lets the designer include them in the main body of another program. The interrupt-driven data transfers give the effect that the serial channels are handled in the background while the main program is executing in the foreground. There are five basic procedures: a serial channel initialization routine and buffer handling routines for the transmit and receive data buffers of each channel. Appendix D-1 shows the entire software listing. Listing line numbers are referenced as each major routing is discussed.

The channel initialization routine (INITIAL 8274), starting with line #203, simply sets each channel into a particular operating mode by loading the command registers of the 8274. In normal operation, once these registers are loaded, they are rarely changed. (Although this example assumes a simple asynchronous operating mode, the concept is easily extended for the byte- and bit-synchronous modes.)



(FOR DETAILED DESCRIPTION ON SDK-86, REFER TO SDK-86 MCS-86 SYSTEM DESIGN KIT ASSEMBLY MANUAL.)

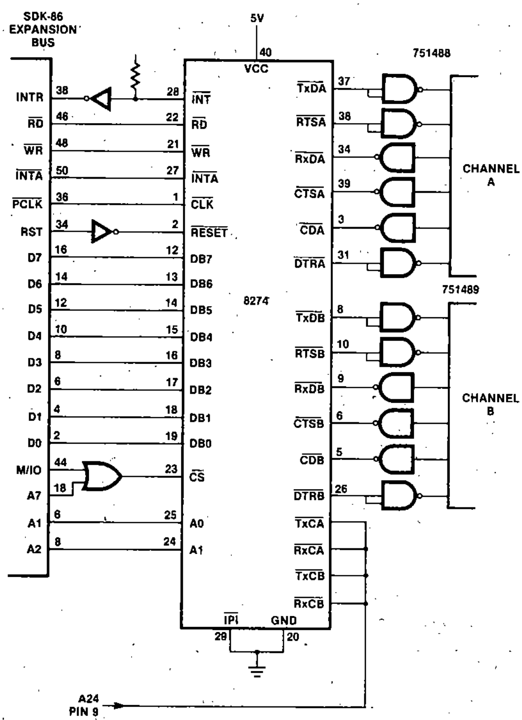


Figure D-1. 8274/SDK-86 Hardware Interface

The channel operating modes are contained in two tables starting with line #163. As the 8274 has only one command register per channel, the remaining seven registers are loaded indirectly through the WR0 (Write Register 0) register. The first byte of each table entry is the register pointer value which is loaded into WR0 and the second byte is the value for that particular register.

The indicated modes set the 8274 for asynchronous operation with data characters 8 bits long, no parity, and 2 stop bits. An X16 baud rate clock is assumed. Also selected is the "interrupt on all RX character" mode with a variable interrupt vector compatible with the 8086/8088. The transmitters are enabled and all model control lines are put in their active state.

In addition to initializing the 8274, this routine also sets up the appropriate interrupt vectors. The 8086 assumes the first 1K bytes of memory contain up to 256 separate interrupt vectors. On the SDK-86 the initial 2K bytes of memory is RAM and therefore must be initialized with the appropriate vectors. (In a prototype system, this initial memory is probably ROM, thus the vector set-up is not needed.) The 8274 supplies up to eight different interrupt vectors. These vectors are developed from internal conditions such as data requests, status changes, or error conditions for each channel. The initialization routine arbitrarily assumes that the initial 8274 vector corresponds to 8086 vector location 80H (memory location 200H). This choice is arbitrary since the 8274 initial vector location is programmable.

Finally, the initialization routine sets up the status and flag in RAM. The meaning and use of these locations are discussed later.

Following the initialization routine are those for the transmit commands (starting with line #268). These commands assume that the host CPU has initialized the publically declared variables for the transmit buffer pointer, TX_POINTER_CHx, and the buffer length, TX_LENGTH_CHx. The transmit command routines simply clear the transmitter empty flag, TX EMPTY CHx, and load the first character of the buffer into the transmitter. It is necessary to load the first character in this manner since transmitter interrupts are generated only when the 8274's transmit data buffer becomes empty. It is the act of becoming empty which generates the interrupt not simply the buffer being empty, thus the transmitter needs one character to start.

The host CPU can monitor the transmitter empty flag, TX_EMPTY_CHx, in order to determine when transmission of the buffer is complete. Obviously, the CPU should only call the command routine after first checking that the empty flag is set.

After returning to the main program, all transmitter data transfers are handled via the transmitter-interrupt service routines starting at lines #360 and #443. These routines start by issuing an End-Of-Interrupt command to the 8274. (This command resets the internal-interrupt controller logic of the 8274 for this particular vector and opens the logic for other internal interrupt requests. The routines next check the length count. If the buffer is completely transmitted, the transmitter empty flag, TX_EMPTY_CHx, is set and a command is issued to the 8274 to reset its interrupt line. Assuming that the buffer is not completely transmitted, the next character is output to the transmitter. In either case, an interrupt return is executed to return to the main CPU program.

The receiver commands start at line #314. Like the transmit commands, it is assumed that the CPU has initialized the receive-buffer-pointer public variable, RX_POINTER_CHx. This variable points to the first location in an empty receive buffer. The command routines clear the receiver ready flag, RX_READY_CHx, and then set the receiver enable bit in the 8274 WR3 register. With the receiver now enabled, any received characters are placed in the receive buffer using interrupt-driven data transfers.

The received data service routines, starting at lines #402 and #485, simply place the received character in the buffer after first issuing the EOI command. The character is then compared to an ASCII CR. An ASCII CR causes the routine to set the receiver ready flag, RX_READY_CHx, and to disable the receiver. The CPU can interrogate this flag to determine when the buffer contains a new line of data. The receive buffer pointer, RX_POINTER_CHx, points to the last received character and the receive counter, RX_COUNTER_CHx, contains the length.

That completes our discussion of the command routines and their associated interrupt service routines. Although not used by the commands, two additional service routines are included for completeness. These routines handle the error and status-change interrupt vectors.

The error service routines, starting at lines #427 and #510, are vectored to if a special receive condition is detected by the 8274. These special receive conditions include parity, receiver overrun, and framing errors. When this vector is generated, the error condition is indicated in RR1 (Read Register 1). The error service routine issues an EO1 command, reads RR1 and places it in the ERROR_MSG_CHx variable, and then issues

a reset error command to the 8274. The CPU can monitor the error message location to detect error conditions. The designer, of course, can supply his own error service routine.

Similarily, the status-change routines (starting lines #386 and #469) are initiated by a change in the modem-control status lines CTS/, CD/, or SYNDET/. (Note that WR2 bit 0 controls whether the 8274 generates interrupts based upon changes in these lines. Our WR2 parameter is such that the 8274 is programmed to ignore changes for these inputs.) The service routines simply read RR0, place its contents in the STATUS_MSG-_CHx variable and then issue a reset external status command. Read Register 0 contains the state of the modem inputs at the point of the last change.

Well, that's it. This application example has presented useful, albeit very simple, routines showing how the 8274 might be used to transmit and receive buffers using an asynchronous serial format. Extensions for byte- or bit-synchronous formats would require no hardware changes due to the highly programmable nature of the 8274's serial formats.

8274 APPLICATION BRIEF PROGRAM

```
MCS-86 MACRO ASSEMBLER    ASYNCB


ISIS-II MCS-86 MACRO ASSEMBLER V2.1 ASSEMBLY OF MODULE ASYNCB
OBJECT MODULE PLACED IN :F1:ASYNCB.OBJ
ASSEMBLER INVOKED BY:  ASM86 :F1:ASYNCB.SRC


LOC  OBJ              LINE     SOURCE

                         1     ;****************************************************************
                         2     ;*                                                              *
                         3     ;*              8274 APPLICATION BRIEF PROGRAM                  *
                         4     ;*                                                              *
                         5     ;*                                                              *
                         6     ;*                                                              *
                         7     ;* THE 8274 IS INITIALIZED FOR SIMPLE ASYNCHRONOUS SERIAL       *
                         8     ;* FORMAT AND VECTORED INTERRUPT-DRIVEN DATA TRANSFERS.         *
                         9     ;* THE INITIALIZATION ROUTINE ALSO LOADS THE 8086'S INTERRUPT   *
                        10     ;* VECTOR TABLE FROM THE CODE SEGMENT INTO LOW RAM ON THE       *
                        11     ;* SDK-86.   THE TRANSMITTER AND RECEIVER ARE LEFT ENABLED.     *
                        12     ;*                                                              *
                        13     ;* FOR TRANSMIT, THE CPU PASSES IN MEMORY THE POINTER OF A      *
                        14     ;* BUFFER TO TRANSMIT AND THE BYTE LENGTH OF THE BUFFER.        *
                        15     ;* THE DATA TRANSFER PROCEED USING INTERRUPT-DRIVEN TRANSFERS.  *
                        16     ;* A STATUS BIT IN MEMORY IS SET WHEN IF BUFFERS IS EMPTY.      *
                        17     ;*                                                              *
                        18     ;* FOR RECEIVE, THE CPU PASSES THE POINTER OF A BUFFER TO FILL. *
                        19     ;* THE BUFFER IS FILLED UNTIL A 'CR_CHR' CHARACTER IS RECEIVED. *
                        20     ;* A STATUS BIT IS SET AND THE CPU MAY READ THE RX POINTER TO   *
                        21     ;* DETERMINE THE LOCATION OF THE LAST CHARACTER.                *
                        22     ;*                                                              *
                        23     ;* ALL ROUTINES ARE ASSUMED TO EXIST IN THE SAME CODE SEGMENT.  *
                        24     ;* CALL'S TO THE SERVICE ROUTINES ARE ASSUMED TO BE "SHORT" OR  *
                        25     ;* INTRASEGMENT (ONLY THE RETURN ADDRESS IP IS ON THE STACK).   *
                        26     ;*                                                              *
                        27     ;*                                                              *
                        28     ;*                                                              *
                        29     ;*                                                              *
                        30     ;****************************************************************
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ                  LINE     SOURCE


                                         NAME    ASYNCB  ;MODULE NAME

                                 ;PUBLIC DECLARATIONS FOR COMMAND ROUTINES

                                         PUBLIC  INITIAL_8274     ;INITIALIZATION ROUTINE
                                         PUBLIC  TX_COMMAND_CHB  ;TX BUFFER COMMAND CHANNEL B
                                         PUBLIC  TX_COMMAND_CHA  ;TX BUFFER COMMAND CHANNEL A
                                         PUBLIC  RX_COMMAND_CHB  ;RX BUFFER COMMAND CHANNEL B
                                         PUBLIC  RX_COMMAND_CHA  ;RX BUFFER COMMAND CHANNEL A

                                 ;PUBLIC DECLARATIONS FOR STATUS VARIABLES

                                         PUBLIC  RX_READY_CHB    ;RX READY FLAG CHB
                                         PUBLIC  RX_READY_CHA    ;RX READY FLAG CHA
                                         PUBLIC  TX_EMPTY_CHB    ;TX EMPTY FLAG CHB
                                         PUBLIC  TX_EMPTY_CHA    ;TX EMPTY FLAG CHA
                                         PUBLIC  RX_COUNT_CHB    ;RX BUFFER COUNTER CHB
                                         PUBLIC  RX_COUNT_CHA    ;RX BUFFER COUNTER CHA
                                         PUBLIC  ERROR_MSG_CHB   ;ERROR FLAG CHB
                                         PUBLIC  ERROR_MSG_CHA   ;ERROR FLAG CHA
                                         PUBLIC  STATUS_MSG_CHB  ;STATUS FLAG CHB
                                         PUBLIC  STATUS_MSG_CHA  ;STATUS FLAG CHA

                                 ;PUBLIC DECLARATIONS FOR VARIABLES PASSED TO THE TRANSMIT
                                 ;AND RECEIVE COMMANDS.

                                         PUBLIC  TX_POINTER_CHB  ;TX BUFFER POINTER FOR CHB
                                         PUBLIC  TX_LENGTH_CHB   ;TX LENGTH OF BUFFER FOR CHB
                                         PUBLIC  TX_POINTER_CHA  ;TX BUFFER POINTER FOR CHA
                                         PUBLIC  TX_LENGTH_CHA   ;TX LENGTH OF BUFFER FOR CHA
                                         PUBLIC  RX_POINTER_CHB  ;RX BUFFER POINTER FOR CHB
                                         PUBLIC  RX_POINTER_CHA  ;RX BUFFER POINTER FOR CHA

                                 ;I/O PORT ASSIGNMENTS

                                 ;CHANNEL A PORT ASSIGNMENTS

 0000                            DATA_PORT_CHA          EQU     0                       ;DATA I/O PORT
 0002                            COMMAND_PORT_CHA       EQU     2                       ;COMMAND PORT
 0002                            STATUS_PORT_CHA        EQU     COMMAND_PORT_CHA        ;STATUS PORT

                                 ;CHANNEL B PORT ASSIGNMENTS

 0004                            DATA_PORT_CHB          EQU     4                       ;DATA I/O PORT
 0006                            COMMAND_PORT_CHB       EQU     6                       ;COMMAND PORT
 0006                            STATUS_PORT_CHB        EQU     COMMAND_PORT_CHB        ;STATUS PORT

                                 ;MISC. SYSTEM EQUATES

 000D                            CR_CHR          EQU     0DH     ;ASCII CR CHARACTER CODE
 0200                            INT_TABLE_BASE  EQU     200H    ;INT. VECTOR BASE ADDRESS
 0500                            CODE_START      EQU     500H    ;START LOCATION FOR CODE

                            +1   $EJECT

                                 ;RAM ASSSIGNMENTS FOR DATA SEGMENT

                                         DATA    SEGMENT

```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ                  LINE    SOURCE

                                  ;VECTOR INTERRUPT TABLE - ASSUME INITIAL 8274 INTERRUPT
                                  ;VECTOR IS NUMBER 80 (0200H).  FOR EACH VECTOR, THE TABLE
                                  ;CONTAINS START LOCATION AND CODE SEGMENT REGISTER VALUE.
                                  ;THE TABLE IS LOADED FROM PROM.

0200                                      ORG     INT_TABLE_BASE

0200 0000                         TX_VECTOR_CHB   DW      0       ;TX INTERRUPT VECTOR FOR CHB
0202 0000                         TX_CS_CHB       DW      0

0204 0000                         STS_VECTOR_CHB  DW      0       ;STATUS INTERRUPT VECTOR FOR CHB
0206 0000                         STS_CS_CHB      DW      0

0208 0000                         RX_VECTOR_CHB   DW      0       ;RX INTERRUPT VECTOR FOR CHB
020A 0000                         RX_CS_CHB       DW      0

020C 0000                         ERR_VECTOR_CHB  DW      0       ;ERROR INTERRUPT VECTOR FOR CHB
020E 0000                         ERR_CS_CHB      DW      0

0210 0000                         TX_VECTOR_CHA   DW      0       ;TX INTERRUPT VECTOR FOR CHA
0212 0000                         TX_CS_CHA       DW      0

0214 0000                         STS_VECTOR_CHA  DW      0       ;STATUS INTERRUPT VECTOR FOR CHA
0216 0000                         STS_CS_CHA      DW      0

0218 0000                         RX_VECTOR_CHA   DW      0       ;RX INTERRUPT VECTOR FOR CHA
021A 0000                         RX_CS_CHA       DW      0

021C 0000                         ERR_VECTOR_CHA  DW      0       ;ERROR INTERRUPT VECTOR FOR CHA
021E 0000                         ERR_CS_CHA      DW      0

                                  ;MISC RAM LOCATIONS FOR CHANNEL STATUS AND POINTERS

                                  ;CHANNEL B POINTERS AND STATUS

0220 0000                         TX_POINTER_CHB  DW      0       ;TX BUFFER POINTER FOR CHB
0222 0000                         TX_LENGTH_CHB   DW      0       ;TX BUFFER LENGTH FOR CHB
0224 0000                         RX_POINTER_CHB  DW      0       ;RX BUFFER POINTER FOR CHB
0226 0000                         RX_COUNT_CHB    DW      0       ;RX LENGTH COUNTER FOR CHB
0228 00                           TX_EMPTY_CHB    DB      0       ;TX DONE FLAG
0229 00                           RX_READY_CHB    DB      0       ;READY FLAG (1 IF CR_CHR RECEIVED, ELSE 0)
022A 00                           STATUS_MSG_CHB  DB      0       ;STATUS CHANGE MESSAGE
022B 00                           ERROR_MSG_CHB   DB      0       ;ERROR STATUS LOCATION (0 IF NO ERROR)

                                  ;CHANNEL A POINTERS AND STATUS

022C 0000                         TX_POINTER_CHA  DW      0       ;TX BUFFER POINTER FOR CHA
022E 0000                         TX_LENGTH_CHA   DW      0       ;TX BUFFER LENGTH FOR CHA
0230 0000                         RX_POINTER_CHA  DW      0       ;RX BUFFER POINTER FOR CHA
0232 0000                         RX_COUNT_CHA    DW      0       ;RX LENGTH COUNTER FOR CHA
0234 00                           TX_EMPTY_CHA    DB      0       ;TX DONE FLAG
0235 00                           RX_READY_CHA    DB      0       ;READY FLAG (1 IF CR_CHR RECEIVED, ELSE 0)
0236 00                           STATUS_MSG_CHA  DB      0       ;STATUS CHANGE MESSAGE
0237 00                           ERROR_MSG_CHA   DB      0       ;ERROR STATUS LOCATION (0 IF NO ERROR)

----                                      DATA    ENDS

                                  $EJECT
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ          SOURCE

----                    ABC     SEGMENT
                        ASSUME  CS:ABC,DS:DATA,SS:DATA
0500                    ORG     CODE_START

              ;*****************************************************************
              ;*                                                               *
              ;*          PARAMETERS FOR CHANNEL INITIALIZATION                *
              ;*                                                               *
              ;*****************************************************************

              ;CHANNEL B PARAMETERS

                        ;WR1 - INTERRUPT ON ALL RX CHR, VARIABLE INT VECTOR, TX INT ENABLE
0500 01       CMDSTRB DB      1,16H
0501 16
                        ;WR2 - INTERRUPT VECTOR
0502 02                 DB      2,(INT_TABLE_BASE/4)
0503 80
                        ;WR3 - RX 8 BITS/CHR, RX DISABLE
0504 03                 DB      3,0C0H
0505 C0
                        ;WR4 - X16 CLOCK, 2 STOP BITS, NO PARITY
0506 04                 DB      4,4CH
0507 4C
                        ;WR5 - DTR ACTIVE, TX 8 BITS/CHR, TX ENABLE, RTS ACTIVE
0508 05                 DB      5,0EAH
0509 EA
                        ;WR6 AND WR7 NOT REQUIRED FOR ASYNC
050A 00                 DB      0,0
050B 00

              ;CHANNEL A PARAMETERS

                        ;WR1 - INTERRUPT ON ALL RX CHR, TX INT ENABLE
050C 01       CMDSTRA DB      1,12H
050D 12
                        ;WR2 - VECTORED INTERRUPT FOR 8086
050E 02                 DB      2,30H
050F 30
                        ;WR3 - RX 8 BITS/CHR, RX DISABLE
0510 03                 DB      3,0C0H
0511 C0
                        ;WR4 - X16 CLOCK, 2 STOP BITS, NO PARITY
0512 04                 DB      4,4CH
0513 4C
                        ;WR5 - DTR ACTIVE, TX 8 BITS/CHR, TX ENABLE, RTS ACTIVE
0514 05                 DB      5,0EAH
0515 EA
                        ;WR6 AND WR7 NOT REQUIRED FOR ASYNC
0516 00                 DB      0,0
0517 00

          +1  $EJECT
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ                LINE     SOURCE


                                 ;START OF COMMAND ROUTINES

                                 ;****************************************************************
                                 ;*                                                              *
                                 ;*      INITIALIZATION COMMAND FOR THE 8274 - THE 8274          *
                                 ;*      IS SETUP ACCORDING TO THE PARAMETERS STORED IN          *
                                 ;*      PROM ABOVE STARTING AT CMSTRB FOR CHANNEL B AND         *
                                 ;*      CMSTRA FOR CHANNEL A.                                   *
                                 ;*                                                              *
                                 ;****************************************************************

0518                             INITIAL_8274:
                                         ;COPY INTERRUPT VECTOR IP AND CS VALUES FROM PROM TO RAM
0518 C70600020806                        MOV    TX_VECTOR_CHB, OFFSET XMTINB    ;TX DATA VECTOR CHB
051E 8C0E0202                            MOV    TX_CS_CHB, CS
0522 C70604023506                        MOV    STS_VECTOR_CHB, OFFSET STAINB   ;STATUS VECTOR CHB
0528 8C0E0602                            MOV    STS_CS_CHB, CS
052C C70608024906                        MOV    RX_VECTOR_CHB, OFFSET RCVINB    ;RX DATA VECTOR CHB
0532 8C0E0A02                            MOV    RX_CS_CHB, CS
0536 C7060C027706                        MOV    ERR_VECTOR_CHB, OFFSET ERRINB   ;ERROR VECTOR CHB
053C 8C0E0A02                            MOV    RX_CS_CHB, CS
0540 C70610028C06                        MOV    TX_VECTOR_CHA, OFFSET XMTINA    ;TX DATA VECTOR CHA
0546 8C0E1202                            MOV    TX_CS_CHA, CS
054A C7061402B906                        MOV    STS_VECTOR_CHA, OFFSET STAINA   ;STATUS VECTOR CHA
0550 8C0E1602                            MOV    STS_CS_CHA, CS
0554 C7061802CD06                        MOV    RX_VECTOR_CHA, OFFSET RCVINA    ;RX DATA VECTOR CHA
055A 8C0E1A02                            MOV    RX_CS_CHA, CS
055E C7061C02FB06                        MOV    ERR_VECTOR_CHA, OFFSET ERRINA   ;ERROR VECTOR CHA
0564 8C0E1E02                            MOV    ERR_CS_CHA, CS

                                 ;COPY SETUP TABLE PARAMETERS INTO 8274

0568 BF0005                              MOV    DI, OFFSET CMDSTRB        ;INITIALIZE CHB
056B BA0600                              MOV    DX, COMMAND_PORT_CHB
056E E82E00                              CALL   SETUP                     ;COPY CHB PARAMETERS
0571 BF0C05                              MOV    DI, OFFSET CMDSTRA        ;INITIALIZE CHA
0574 BA0200                              MOV    DX, COMMAND_PORT_CHA
0577 E82500                              CALL   SETUP                     ;COPY CHA PARAMETERS

                                 ;INITIALIZE STATUS BYTES AND FLAGS

057A B80000                              MOV    AX, 0
057D A22B02                              MOV    ERROR_MSG_CHB, AL         ;CLEAR ERROR FLAG CHB
0580 A23702                              MOV    ERROR_MSG_CHA, AL         ;CLEAR ERROR FLAG CHA
0583 A22A02                              MOV    STATUS_MSG_CHB, AL        ;CLEAR STATUS FLAG CHB
0586 A23602                              MOV    STATUS_MSG_CHA, AL        ;CLEAR STATUS FLAG CHA
0589 A32602                              MOV    RX_COUNT_CHB, AX          ;CLEAR RX COUNTER CHB
058C A33202                              MOV    RX_COUNT_CHA, AX          ;CLEAR RX COUNTER CHA
058F B001                                MOV    AL, 1
0591 A22902                              MOV    RX_READY_CHB, AL          ;SET RX DONE FLAG CHB
0594 A23502                              MOV    RX_READY_CHA, AL          ;SET RX DONE FLAG CHA
0597 A22802                              MOV    TX_EMPTY_CHB, AL          ;SET TX DONE FLAG CHB
059A A23402                              MOV    TX_EMPTY_CHA, AL          ;SET TX DONE FLAG CHA
059D FB                                  STI                              ;ENABLE INTERRUPTS
059E C3                                  RET                              ;RETURN - DONE WITH SETUP

059F 8A05                        SETUP:  MOV    AL, [DI]                  ;PARAMETER COPYING ROUTINE
05A1 3C00                                CMP    AL, 0
05A3 7404                                JE     DONE
```

```
LOC  OBJ          SOURCE

05A5 EE                   OUT     DX, AL                  ;OUTPUT PARAMETER
05A6 47                   INC     DI                      ;POINT AT NEXT PARAMETER
05A7 EBF6                 JMP     SETUP                   ;GO LOAD IT
05A9 C3           DONE:   RET                             ;DONE - SO RETURN

             +1   $EJECT

                  ;*****************************************************************
                  ;*                                                               *
                  ;*    TX CHANNEL B COMMAND ROUTINE - ROUTINE IS CALLED TO        *
                  ;*    TRANSMIT A BUFFER.  THE BUFFER STARTING ADDRESS,           *
                  ;*    TX_POINTER_CHB, AND THE BUFFER LENGTH, TX_LENGTH_CHB,      *
                  ;*    MUST BE INITIALIZED BY THE CALLING PROGRAM.                *
                  ;*    BOTH ITEMS ARE WORD VARIABLES.                             *
                  ;*                                                               *
                  ;*****************************************************************

05AA              TX_COMMAND_CHB:
05AA 50                   PUSH    AX              ;SAVE REGISTERS
05AB 57                   PUSH    DI
05AC 52                   PUSH    DX
05AD C606280200           MOV     TX_EMPTY_CHB, 0 ;CLEAR EMPTY FLAG
05B2 BA0400               MOV     DX, DATA_PORT_CHB       ;SETUP PORT POINTER
05B5 8B3E2002             MOV     DI, TX_POINTER_CHB      ;GET TX BUFFER POINTER CHB
05B9 8A05                 MOV     AL, [DI]        ;GET FIRST CHARACTER TO TX
05BB EE                   OUT     DX, AL          ;OUTPUT IT TO 8274 TO GET IT STARTED
05BC 5A                   POP     DX
05BD 5F                   POP     DI
05BE 58                   POP     AX
05BF C3                   RET                     ;RETURN

                  ;*****************************************************************
                  ;*                                                               *
                  ;*    TX CHANNEL A COMMAND ROUTINE - ROUTINE IS CALLED TO        *
                  ;*    TRANSMIT A BUFFER.  THE BUFFER STARTING ADDRESS,           *
                  ;*    TX_POINTER_CHA, AND THE BUFFER LENGTH, TX_LENGTH_CHA,      *
                  ;*    MUST BE INITIALIZED BY THE CALLING PROGRAM.                *
                  ;*    BOTH ITEMS ARE WORD VARIABLES.                             *
                  ;*                                                               *
                  ;*****************************************************************

05C0              TX_COMMAND_CHA:
05C0 50                   PUSH    AX              ;SAVE REGISTERS
05C1 57                   PUSH    DI
05C2 52                   PUSH    DX
05C3 C606340200           MOV     TX_EMPTY_CHA, 0 ;CLEAR EMPTY FLAG
05C8 BA0000               MOV     DX, DATA_PORT_CHA       ;SETUP PORT POINTER
05CB 8B3E2C02             MOV     DI, TX_POINTER_CHA      ;GET TX BUFFER POINTER CHA
05CF 8A05                 MOV     AL, [DI]        ;GET FIRST CHARACTER TO TX
05D1 EE                   OUT     DX, AL          ;OUTPUT IT TO 8274 TO GET IT STARTED
05D2 5A                   POP     DX
05D3 5F                   POP     DI
05D4 58                   POP     AX
05D5 C3                   RET                     ;RETURN

                  ;*****************************************************************
                  ;*                                                               *
                  ;*    RX COMMAND FOR CHANNEL B - THE CALLING ROUTINE MUST        *
                  ;*    INITIALIZE RX_POINTER_CHB TO POINT AT THE RECEIVE          *
                  ;*    BUFFER BEFORE CALLING THIS ROUTINE.                        *
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ                   SOURCE

                           ;*                                                              *
                           ;****************************************************************

05D6                       RX_COMMAND_CHB:
05D6 50                            PUSH    AX              ;SAVE REGISTERS
05D7 52                            PUSH    DX
05D8 C606290200                    MOV     RX_READY_CHB, 0 ;CLEAR RX READY FLAG
05DD C70626020000                  MOV     RX_COUNT_CHB, 0 ;CLEAR RX COUNTER
05E3 BA0600                        MOV     DX, COMMAND_PORT_CHB    ;POINT AT COMMAND PORT
05E6 B003                          MOV     AL, 3           ;SET UP FOR WR3
05E8 EE                            OUT     DX, AL
05E9 B0C1                          MOV     AL, 0C1H        ;WR3 - 8 BITS/CHR, ENABLE RX
05EB EE                            OUT     DX, AL
05EC 5A                            POP     DX
05ED 58                            POP     AX
05EE C3                            RET                     ;RETURN

                           ;****************************************************************
                           ;*                                                              *
                           ;*    RX COMMAND FOR CHANNEL A - THE CALLING ROUTINE MUST       *
                           ;*    INITIALIZE RX_POINTER_CHA TO POINT AT THE RECEIVE         *
                           ;*    BUFFER BEFORE CALLING THIS ROUTINE.                       *
                           ;*                                                              *
                           ;****************************************************************

05EF                       RX_COMMAND_CHA:
05EF 50                            PUSH    AX              ;SAVE REGISTERS
05F0 52                            PUSH    DX
05F1 C606350200                    MOV     RX_READY_CHA, 0 ;CLEAR RX READY FLAG
05F6 C70632020000                  MOV     RX_COUNT_CHA, 0 ;CLEAR RX COUNTER
05FC BA0200                        MOV     DX, COMMAND_PORT_CHA    ;POINT AT COMMAND PORT
05FF B003                          MOV     AL, 3           ;SET UP FOR WR3
0601 EE                            OUT     DX, AL
0602 B0C1                          MOV     AL, 0C1H        ;WR3 - 8 BITS/CHR, ENABLE RX
0604 EE                            OUT     DX, AL
0605 5A                            POP     DX
0606 58                            POP     AX
0607 C3                            RET                     ;RETURN

                        +1 $EJECT

                           ;****************************************************************
                           ;*                                                              *
                           ;*          START OF INTERRUPT SERVICE ROUTINES                 *
                           ;*                                                              *
                           ;****************************************************************

                           ;CHANNEL B TRANSMIT DATA SERVICE ROUTINE

0608 52                    XMTINB: PUSH    DX              ;SAVE REGISTERS
0609 57                            PUSH    DI
060A 50                            PUSH    AX
060B E80201                        CALL    EOI             ;SEND EOI COMMAND TO 8274
060E FF062002                      INC     TX_POINTER_CHB  ;POINT TO NEXT CHARACTER
0612 FF0E2202                      DEC     TX_LENGTH_CHB   ;DEC LENGTH COUNTER
0616 740E                          JE      XIB             ;TEST IF DONE
0618 BA0400                        MOV     DX, DATA_PORT_CHB       ;NOT DONE - GET NEXT CHARACTER
061B 8B3E2002                      MOV     DI, TX_POINTER_CHB
061F 8A05                          MOV     AL, [DI]        ;PUT CHARACTER IN AL
0621 EE                            OUT     DX, AL          ;OUTPUT IT TO 8274
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ             LINE     SOURCE

0622 58                              POP     AX                   ;RESTORE REGISTERS
0623 5F                              POP     DI
0624 5A                              POP     DX
0625 CF                              IRET                         ;RETURN TO FOREGROUND
0626 BA0600                   XIB:   MOV     DX, COMMAND_PORT_CHB    ;ALL CHARACTERS HAVE BEEN SEND
0629 B028                            MOV     AL, 28H              ;RESET TRANSMITTER INTERRUPT PENDING
062B EE                              OUT     DX, AL
062C C606280201                      MOV     TX_EMPTY_CHB, 1 ;DONE - SO SET TX EMPTY FLAG CHB
0631 58                              POP     AX                   ;RESTORE REGISTERS
0632 5F                              POP     DI
0633 5A                              POP     DX
0634 CF                              IRET                         ;RETURN TO FOREGROUND

                              ;CHANNEL B STATUS CHANGE SERVICE ROUTINE

0635 52                       STAINB: PUSH   DX                   ;SAVE REGISTERS
0636 57                              PUSH    DI
0637 50                              PUSH    AX
0638 E8D500                          CALL    EOI                  ;SEND EOI COMMAND TO 8274
063B BA0600                          MOV     DX, COMMAND_PORT_CHB
063E EC                              IN      AL, DX               ;READ RR0
063F A22A02                          MOV     STATUS_MSG_CHB, AL      ;PUT RR0 IN STATUS MESSAGE
0642 B010                            MOV     AL, 10H              ;SEND RESET STATUS INT COMMAND TO 8274
0644 EE                              OUT     DX, AL
0645 58                              POP     AX                   ;RESTORE REGISTERS
0646 5F                              POP     DI
0647 5A                              POP     DX
0648 CF                              IRET

                              ;CHANNEL B RECEIVED DATA SERVICE ROUTINE

0649 52                       RCVINB: PUSH   DX                   ;SAVE REGISTERS
064A 57                              PUSH    DI
064B 50                              PUSH    AX
064C E8C100                          CALL    EOI                  ;SEND EOI COMMAND TO 8274
064F 8B3E2402                        MOV     DI, RX_POINTER_CHB      ;GET RX CHB BUFFER POINTER
0653 BA0400                          MOV     DX, DATA_PORT_CHB
0656 EC                              IN      AL, DX               ;READ CHARACTER
0657 8805                            MOV     [DI], AL             ;STORE IN BUFFER
0659 FF062402                        INC     RX_POINTER_CHB  ;BUMP THE BUFFER POINTER
065D FF062602                        INC     RX_COUNT_CHB    ;BUMP THE COUNTER
0661 3C0D                            CMP     AL, CR_CHR      ;TEST IF LAST CHARACTER TO BE RECEIVED?
0663 750E                            JNE     RIB
0665 C606290201                      MOV     RX_READY_CHB, 1 ;YES, SET READY FLAG
066A BA0600                          MOV     DX, COMMAND_PORT_CHB    ;POINT AT COMMAND PORT
066D B003                            MOV     AL, 3                ;POINT AT WR3
066F EE                              OUT     DX, AL
0670 B0C0                            MOV     AL, 0C0H             ;DISABLE RX
0672 EE                              OUT     DX, AL
0673 58                       RIB:   POP     AX                   ;EITHER WAY, RESTORE REGISTERS
0674 5F                              POP     DI
0675 5A                              POP     DX
0676 CF                              IRET                         ;RETURN TO FOREGROUND

                              ;CHANNEL B ERROR SERVICE ROUTINE

0677 52                       ERRINB: PUSH   DX                   ;SAVE REGISTERS
0678 50                              PUSH    AX
0679 E89400                          CALL    EOI                  ;SEND EOI COMMAND TO 8274
067C BA0600                          MOV     DX, COMMAND_PORT_CHB
```

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ                LINE     SOURCE

067F B001                                MOV     AL, 1            ;POINT AT RR1
0681 EE                                  OUT     DX, AL
0682 EC                                  IN      AL, DX           ;READ RR1
0683 A22B02                              MOV     ERROR_MSG_CHB, AL      ;SAVE IT IN ERROR FLAG
0686 B030                                MOV     AL, 30H          ;SEND RESET ERROR COMMAND TO 8274
0688 EE                                  OUT     DX, AL
0689 58                                  POP     AX               ;RESTORE REGISTERS
068A 5A                                  POP     DX
068B CF                                  IRET                     ;RETURN TO FOREGROUND

                                 ;CHANNEL A TRANSMIT DATA SERVICE ROUTINE

068C 52                          XMTINA: PUSH    DX               ;SAVE REGISTERS
068D 57                                  PUSH    DI
068E 50                                  PUSH    AX
068F E87E00                              CALL    EOI              ;SEND EOI COMMAND TO 8274
0692 FF062C02                            INC     TX_POINTER_CHA   ;POINT TO NEXT CHARACTER
0696 FF0E2E02                            DEC     TX_LENGTH_CHA    ;DEC LENGTH COUNTER
069A 740E                                JE      XIA              ;TEST IF DONE
069C BA0000                              MOV     DX, DATA_PORT_CHA      ;NOT DONE - GET NEXT CHARACTER
069F 8B3E2C02                            MOV     DI, TX_POINTER_CHA
06A3 8A05                                MOV     AL, [DI]         ;PUT CHARACTER IN AL
06A5 EE                                  OUT     DX, AL           ;OUTPUT IT TO 8274
06A6 58                                  POP     AX               ;RESTORE REGISTERS
06A7 5F                                  POP     DI
06A8 5A                                  POP     DX
06A9 CF                                  IRET                     ;RETURN TO FOREGROUND
06AA BA0200                      XIA:    MOV     DX, COMMAND_PORT_CHA   ;ALL CHARACTERS HAVE BEEN SEND
06AD B028                                MOV     AL, 28H          ;RESET TRANSMITTER INTERRUPT PENDING
06AF EE                                  OUT     DX, AL
06B0 C606340201                          MOV     TX_EMPTY_CHA, 1 ;DONE - SO SET TX EMPTY FLAG CHB
06B5 58                                  POP     AX               ;RESTORE REGISTERS
06B6 5F                                  POP     DI
06B7 5A                                  POP     DX
06B8 CF                                  IRET                     ;RETURN TO FOREGROUND

                                 ;CHANNEL A STATUS CHANGE SERVICE ROUTINE

06B9 52                          STAINA: PUSH    DX               ;SAVE REGISTERS
06BA 57                                  PUSH    DI
06BB 50                                  PUSH    AX
06BC E85100                              CALL    EOI              ;SEND EOI COMMAND TO 8274
06BF BA0200                              MOV     DX, COMMAND_PORT_CHA
06C2 EC                                  IN      AL, DX           ;READ RR0
06C3 A23602                              MOV     STATUS_MSG_CHA, AL     ;PUT RR0 IN STATUS MESSAGE
06C6 B010                                MOV     AL, 10H          ;SEND RESET STATUS INT COMMAND TO 8274
06C8 EE                                  OUT     DX, AL
06C9 58                                  POP     AX               ;RESTORE REGISTERS
06CA 5F                                  POP     DI
06CB 5A                                  POP     DX
06CC CF                                  IRET

                                 ;CHANNEL A RECEIVED DATA SERVICE ROUTINE

06CD 52                          RCVINA: PUSH    DX               ;SAVE REGISTERS
06CE 57                                  PUSH    DI
06CF 50                                  PUSH    AX
06D0 E83D00                              CALL    EOI              ;SEND EOI COMMAND TO 8274
06D3 8B3E3002                            MOV     DI, RX_POINTER_CHA     ;GET RX CHA BUFFER POINTER
06D7 BA0000                              MOV     DX, DATA_PORT_CHA
```


210311-001

```
MCS-86 MACRO ASSEMBLER    ASYNCB

LOC  OBJ            LINE    SOURCE

06DA EC                             IN      AL, DX          ;READ CHARACTER
06DB 8805                           MOV     [DI], AL        ;STORE IN BUFFER
06DD FF063002                       INC     RX_POINTER_CHA  ;BUMP THE BUFFER POINTER
06E1 FF063202                       INC     RX_COUNT_CHA    ;BUMP THE COUNTER
06E5 3C0D                           CMP     AL, CR_CHR      ;TEST IF LAST CHARACTER TO BE RECEIVED?
06E7 750E                           JNE     RIA
06E9 C606350201                     MOV     RX_READY_CHA, 1 ;YES, SET READY FLAG
06EE BA0200                         MOV     DX, COMMAND_PORT_CHA    ;POINT AT COMMAND PORT
06F1 B003                           MOV     AL, 3           ;POINT AT WR3
06F3 EE                             OUT     DX, AL
06F4 B0C0                           MOV     AL, 0C0H        ;DISABLE RX
06F6 EE                             OUT     DX, AL
06F7 58                     RIA:    POP     AX              ;EITHER WAY, RESTORE REGISTERS
06F8 5F                             POP     DI
06F9 5A                             POP     DX
06FA CF                             IRET                    ;RETURN TO FOREGROUND

                            ;CHANNEL A ERROR SERVICE ROUTINE

06FB 52                     ERRINA: PUSH    DX              ;SAVE REGISTERS
06FC 50                             PUSH    AX
06FD E81000                         CALL    EOI             ;SEND EOI COMMAND TO 8274
0700 BA0200                         MOV     DX, COMMAND_PORT_CHA
0703 B001                           MOV     AL, 1           ;POINT AT RR1
0705 EE                             OUT     DX, AL
0706 EC                             IN      AL, DX          ;READ RR1
0707 A23702                         MOV     ERROR_MSG_CHA, AL       ;SAVE IT IN ERROR FLAG
070A B030                           MOV     AL, 30H         ;SEND RESET ERROR COMMAND TO 8274
070C EE                             OUT     DX, AL
070D 58                             POP     AX              ;RESTORE REGISTERS
070E 5A                             POP     DX
070F CF                             IRET                    ;RETURN TO FOREGROUND

                            ;END-OF-INTERRUPT ROUTINE - SENDS EOI COMMAND TO 8274.
                            ;  THIS COMMAND MUST ALWAYS TO ISSUED ON CHANNEL A.

0710 50                     EOI:    PUSH    AX              ;SAVE REGISTERS
0711 52                             PUSH    DX
0712 BA0200                         MOV     DX, COMMAND_PORT_CHA    ;ALWAYS FOR CHANNEL A !!!
0715 B038                           MOV     AL, 38H
0717 EE                             OUT     DX, AL
0718 5A                             POP     DX
0719 58                             POP     AX
071A C3                             RET

                            ;END OF CODE ROUTINE

----                                ABC     ENDS
                                    END

ASSEMBLY COMPLETE, NO ERRORS FOUND
```

REFERENCES

1. 8274 Multiprotocol Serial Controller (MPSC) Data Sheet, Intel Corporation, California, 1980.
2. Basics of Data Communication, Electronics Book Series, McGraw-Hill, New York, 1976.
3. Telecommunications and the Computer, J. Martin, Prentice-Hall, New Jersey, 1976.
4. Technical Aspects of Data Communications, J. McNamara, DEC Press, Massachusetts, 1977.
5. Miscellaneous Data Communications Standards —EIA RS-232-C, EIA RS-422, EIA RS-423, EIA Standard Sales, Washington, D.C.